DISSERTATION

PROOF THEORY AND ALGORITHMS FOR ANSWER SET PROGRAMMING

VORGELEGT VON

MARTIN GEBSER

ZUR ERLANGUNG DES AKADEMISCHEN GRADES
DOKTOR DER NATURWISSENSCHAFTEN (DR. RER. NAT.)
IN DER WISSENSCHAFTSDISZIPLIN
"WISSENSVERARBEITUNG UND INFORMATIONSSYSTEME"

EINGEREICHT AN DER
MATHEMATISCH-NATURWISSENSCHAFTLICHEN FAKULTÄT
UNIVERSITÄT POTSDAM

ANGEFERTIGT AM
INSTITUT FÜR INFORMATIK
PROFESSUR FÜR WISSENSVERARBEITUNG UND INFORMATIONSSYSTEME

BETREUT VON
PROF. DR. TORSTEN SCHAUB

BEGUTACHTET VON
PROF. DR. GERHARD BREWKA
PROF. DR. TOMI JANHUNEN
PROF. DR. TORSTEN SCHAUB

POTSDAM, IM OKTOBER 2011

Published online at the
Institutional Repository of the University of Potsdam:
URL http://opus.kobv.de/ubp/volltexte/2011/5542/
URN urn:nbn:de:kobv:517-opus-55425
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-55425

Zusammenfassung

Antwortmengenprogrammierung (engl. Answer Set Programming; ASP) ist ein Paradigma zum deklarativen Problemlösen, wobei Problemstellungen durch logische Programme beschrieben werden, sodass bestimmte Modelle, Antwortmengen genannt, zu Lösungen korrespondieren. Die zunehmenden praktischen Anwendungen von ASP verlangen nach performanten Werkzeugen zum Lösen komplexer Problemstellungen.

ASP integriert diverse Konzepte aus verwandten Bereichen. Insbesondere sind automatisierte Techniken für die Suche nach Antwortmengen durch Verfahren zum Lösen des aussagenlogischen Erfüllbarkeitsproblems (engl. Boolean Satisfiability; SAT) inspiriert. Letztere beruhen auf soliden beweistheoretischen Grundlagen, wohingegen es für ASP kaum formale Systeme gibt, um Lösungsmethoden einheitlich zu beschreiben und miteinander zu vergleichen. Weiterhin basiert der Erfolg moderner Verfahren zum Lösen von SAT entscheidend auf fortgeschrittenen Suchtechniken, die in gängigen Methoden zur Antwortmengenberechnung nicht etabliert sind.

Diese Arbeit entwickelt beweistheoretische Grundlagen und fortgeschrittene Suchtechniken im Kontext der Antwortmengenberechnung. Unsere formalen Beweissysteme ermöglichen die Charakterisierung, den Vergleich und die Analyse vorhandener Lösungsmethoden für ASP. Außerdem entwerfen wir moderne Verfahren zum Lösen von ASP, die fortgeschrittene Suchtechniken aus dem SAT-Bereich integrieren und erweitern. Damit trägt diese Arbeit sowohl zum tieferen Verständnis von Lösungsmethoden für ASP und ihrer Beziehungen untereinander als auch zu ihrer Verbesserung durch die Erschließung fortgeschrittener Suchtechniken bei.

Die zentrale Idee unseres Ansatzes besteht darin, Atome und komposite Konstrukte innerhalb von logischen Programmen gleichermaßen mit aussagenlogischen Variablen zu assoziieren. Dies ermöglicht die Isolierung fundamentaler Inferenzschritte, die wir in formalen Charakterisierungen von Lösungsmethoden für ASP selektiv miteinander kombinieren können. Darauf aufbauend zeigen wir, dass unterschiedliche Einschränkungen von Fallunterscheidungen zwangsläufig zu exponentiellen Effizienzunterschieden zwischen den charakterisierten Methoden führen. Wir generalisieren unseren beweistheoretischen Ansatz auf logische Programme mit erweiterten Sprachkonstrukten und weisen analytisch nach, dass das Treffen bzw. Unterlassen von Fallunterscheidungen auf solchen Konstrukten ebenfalls exponentielle Effizienzunterschiede bedingen kann.

Die zuvor beschriebenen fundamentalen Inferenzschritte nutzen wir zur Extraktion inhärenter Bedingungen, denen Antwortmengen genügen müssen. Damit schaffen wir eine Grundlage für den Entwurf moderner Lösungsmethoden für ASP, die fortgeschrittene, ursprünglich für SAT konzipierte, Suchtechniken mit einschließen und darüber hinaus einen transparenten Technologietransfer zwischen Verfahren zum Lösen von ASP und SAT erlauben. Neben der Suche nach einer Antwortmenge behandeln wir ihre Aufzählung, sowohl für gesamte Antwortmengen als auch für Projektionen auf ein Subvokabular. Hierfür entwickeln wir neuartige Methoden, die wiederholungsfreies Aufzählen in polynomiellem Platz ermöglichen, ohne die Suche zu beeinflussen und ggf. zu behindern, bevor Antwortmengen berechnet wurden.

Abstract

Answer Set Programming (ASP) is an emerging paradigm for declarative programming, in which a computational problem is specified by a logic program such that particular models, called answer sets, match solutions. ASP faces a growing range of applications, demanding for high-performance tools able to solve complex problems.

ASP integrates ideas from a variety of neighboring fields. In particular, automated techniques to search for answer sets are inspired by Boolean Satisfiability (SAT) solving approaches. While the latter have firm proof-theoretic foundations, ASP lacks formal frameworks for characterizing and comparing solving methods. Furthermore, sophisticated search patterns of modern SAT solvers, successfully applied in areas like, e.g., model checking and verification, are not yet established in ASP solving.

We address these deficiencies by, for one, providing proof-theoretic frameworks that allow for characterizing, comparing, and analyzing approaches to answer set computation. For another, we devise modern ASP solving algorithms that integrate and extend state-of-the-art techniques for Boolean constraint solving. We thus contribute to the understanding of existing ASP solving approaches and their interconnections as well as to their enhancement by incorporating sophisticated search patterns.

The central idea of our approach is to identify atomic as well as composite constituents of a propositional logic program with Boolean variables. This enables us to describe fundamental inference steps, and to selectively combine them in proof-theoretic characterizations of various ASP solving methods. In particular, we show that different concepts of case analyses applied by existing ASP solvers implicate mutual exponential separations regarding their best-case complexities. We also develop a generic proof-theoretic framework amenable to language extensions, and we point out that exponential separations can likewise be obtained due to case analyses on them.

We further exploit fundamental inference steps to derive Boolean constraints characterizing answer sets. They enable the conception of ASP solving algorithms including search patterns of modern SAT solvers, while also allowing for direct technology transfers between the areas of ASP and SAT solving. Beyond the search for one answer set of a logic program, we address the enumeration of answer sets and their projections to a subvocabulary, respectively. The algorithms we develop enable repetition-free enumeration in polynomial space without being intrusive, i.e., they do not necessitate any modifications of computations before an answer set is found.

Our approach to ASP solving is implemented in *clasp*, a state-of-the-art Boolean constraint solver that has successfully participated in recent solver competitions. Although we do here not address the implementation techniques of *clasp* or all of its features, we present the principles of its success in the context of ASP solving.

Acknowledgments

Before I start to forget anyone, note that these acknowledgments are the last piece written after improving parts of this thesis over and over again. The journey does not end here, and certainly a lot of what is written may be done much simpler and elegant in a not so far future. Therefore, I first of all thank the considerate reader for paying attention: for you, I spent so much time and effort. Special thanks go to the external referees, Gerhard Brewka and Tomi Janhunen, for accepting the charge to review a draft of this thesis and for providing me with many constructive comments.

Moreover, I thank my supervisor, Torsten Schaub, for his constant support, patience, and inspiration. We jointly made a stretch of way, and it is amazing to see how far we came up to now, already. All this would not have been possible without the brilliant work of current and former colleagues, including: Benjamin Andres, Sylvain Blachon, Christian Drescher, Steve Dworschak, Torsten Grote, Roland Kaminski, Benjamin Kaufmann, Arne König, Thomas Linke, André Neumann, Max Ostrowski, Orkunt Sabuncu, Marius Schneider, Sven Thiele, and Philippe Veber. Thanks for your invaluable contributions!

I also want to thank Chitta Baral, Martin Brain, François Coste, Marina De Vos, Jim Delgrande, Thomas Eiter, Carito Guziolowski, Antti Hyvärinen, Tomi Janhunen, Matti Järvisalo, Tommi Junttila, Thomas Krennwallner, Joohyung Lee, Yuliya Lierler, Vladimir Lifschitz, Jacques Nicolas, Ilkka Niemelä, Johannes Oetsch, Emilia Oikarinen, Jörg Pührer, Anne Siegel, Hans Tompits, and Stefan Woltran for giving me the chance to visit them.

Last but not least, I thank my family and friends for being around and supporting me all the time. Accomplishing this thesis would not have been possible without them.

The work on this thesis has been partially supported by the German Science Foundation (DFG) under grants SCHA 550/8-1/2.

Contents

Chapter 1

Introduction

Answer Set Programming (ASP) [12] is an emerging framework for knowledge representation and reasoning.[1] It integrates ideas from the fields of Non-Monotonic Reasoning (NMR) [31], Boolean Satisfiability (SAT) [21] and Constraint Programming (CP) [201], Deductive Databases (Datalog) [1, 215], and Logic Programming (LP) [51, 172]. Originally conceived as a declarative semantics for logic programs including negation as failure [119, 120], ASP has developed into a declarative programming paradigm offering rich yet easy modeling languages [82, 158, 211] along with powerful off-the-shelf reasoning engines [3, 26, 52, 95, 123, 139, 140, 141, 154, 158, 160, 167, 168, 171, 209, 218]. It meanwhile has been used in numerous application areas, such as product configuration [210], decision support for NASA shuttle controllers [190], compiler superoptimization [25], composition of Renaissance music [22], knowledge management [129], synthesis of multiprocessor systems [134], reasoning tools in systems biology [62, 80, 115], and many more.[2]

Declarative programming in ASP follows the principal workflow shown in Figure 1.1. A given computational problem is represented by a logic program such that particular models of the program, called answer sets, match the problem's solutions. ASP allows for solving all computational problems contained in NP (and Σ_2^P) in a uniform way [206], offering more succinct problem representations than classical logic [136, 164]. To facilitate knowledge representation, problem modeling and answer set computation usually consist of two parts each. As illustrated in Figure 1.2, a problem encoding represents domain knowledge in terms of (schematic) rules including universally quantified first-order variables, while a problem instance is typically provided by facts. The first component of an ASP system, a grounder, combines encoding and instance into a propositional logic program, and the second component, a solver, searches for answer sets. Hence, for solving a problem in ASP, a user ought to provide an encoding of domain knowledge and facts describing an instance, while a general-purpose ASP system performs remaining computations.

For illustration, let us consider two examples from [82]: N-coloring and Hamiltonian cycle. Both problems apply to graphs, like the one shown in Figure 1.3. This graph embodies an instance that can be represented by the following facts (written in the input language of ASP grounder *gringo* [82, 90, 91], adopting the shorthands "`..`" and "`;`" from the input language of *lparse* [211] to bundle arguments of predicates):

[1] See also [5, 117, 118, 161, 176, 186] for introductions to the methodology of ASP.

[2] See also `http://www.cs.uni-potsdam.de/~torsten/asp` for an overview of ASP's applications.

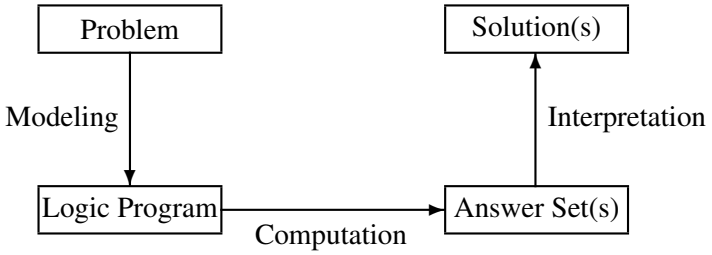


Figure 1.1: Declarative problem solving in answer set programming.

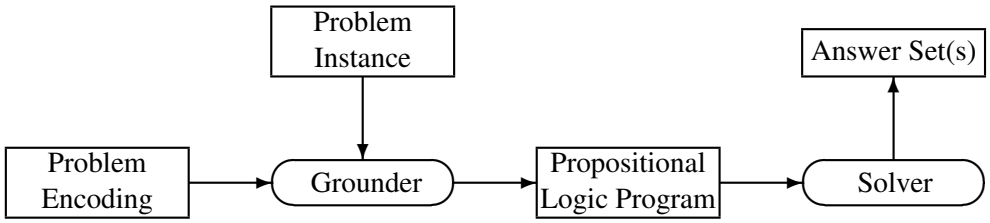


Figure 1.2: Basic architecture of answer set programming systems.

```
% Vertices
vertex(1..6).
% (Directed) Edges
edge(1,2;3;4).  edge(2,4;5;6).  edge(3,1;4;5).
edge(4,1;2).    edge(5,3;4;6).  edge(6,2;3;5).
```

Given such an instance, we need to provide encodings of domain knowledge.

In N-coloring, we require adjacent vertices to be colored differently. Using 3 colors, in the input language of *gringo*, such domain knowledge can be represented as follows:

```
% Default
#const n = 3.
% Generate
1 { color(X,1..n) } 1 :- vertex(X).
% Test
:- edge(X,Y), color(X,C), color(Y,C).
```

The `Generate` rule expresses that exactly one color must be assigned to each vertex, and the `Test` rule eliminates colorings assigning the same color to adjacent vertices. The combination of encoding and instance yields an answer set containing the atoms `color(1,1)`, `color(2,2)`, `color(3,2)`, `color(4,3)`, `color(5,1)`, and `color(6,3)`. That is, vertices `1` and `5`, `2` and `3`, as well as `4` and `6` share a color. Since these vertex pairs are not interconnected by any (directed) edge, the answer set represents a solution of 3-coloring for the given graph.

In Hamiltonian cycle, we are interested in a closed path that visits each vertex exactly once. In the input language of *gringo*, this knowledge can be represented as follows:

```
% Generate
1 { cycle(X,Y) : edge(X,Y) } 1 :- vertex(X).
1 { cycle(X,Y) : edge(X,Y) } 1 :- vertex(Y).
```

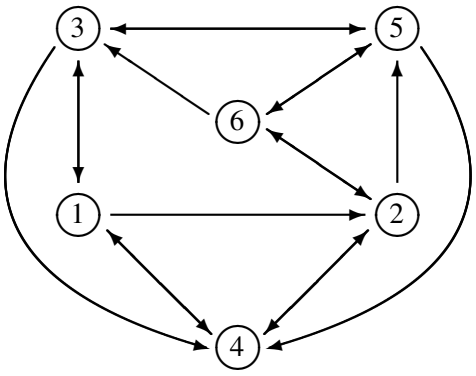


Figure 1.3: A directed graph with six vertices and seventeen edges.

```
% Define
reached(Y) :- cycle(1,Y).
reached(Y) :- cycle(X,Y), reached(X).
% Test
:- vertex(Y), not reached(Y).
```

The `Generate` rules express that each vertex must have exactly one outgoing and exactly one incoming edge. The `Define` rules determine the set of vertices that can be reached from vertex `1` (w.l.o.g., assumed to exist in every graph for the sake of simplicity) by following edges contributing to the cycle. Finally, the `Test` rule eliminates answer set candidates such that some vertex is not reached (starting from `1`). The combination of encoding and instance yields an answer set containing the atoms `cycle(1,2)`, `cycle(2,6)`, `cycle(6,5)`, `cycle(5,3)`, `cycle(3,4)`, and `cycle(4,1)`. These atoms represent the Hamiltonian cycle $(1, 2, 6, 5, 3, 4, 1)$ of the given graph.

Although the input languages of ASP systems include first-order variables, this thesis limits the attention to propositional logic programs and the solving component searching for answer sets (cf. Figure 1.2).

At the propositional level, ASP can be viewed as a relative of SAT. The *Davis-Putnam-Logemann-Loveland* procedure (DPLL) [42, 43], developed about 50 years ago, constitutes a well-known traditional approach to SAT solving. The basic outline of DPLL is as follows:

loop

  propagate // compute deterministic consequences

  if no conflict **then**

    if all variables assigned **then return** variable assignment

    else *decide* // non-deterministically assign some literal

  else

    if top-level conflict **then return** unsatisfiable

    else

      backtrack // undo assignments made after last decision

      flip // assign complement of last decision literal

The basic idea is to combine deterministic (unit) propagation with systematic backtracking, in case of a conflict flipping the last non-deterministically assigned literal.

The search pattern of modern (industrial) SAT solvers is referred to as *Conflict-Driven Clause Learning* (CDCL) [56, 179, 185], whose basic outline is as follows:

```
loop
    propagate                          // compute deterministic consequences
    if no conflict then
        if all variables assigned then return variable assignment
        else decide                    // non-deterministically assign some literal
    else
        if top-level conflict then return unsatisfiable
        else
            analyze                    // analyze conflict and add a conflict constraint
            backjump                   // undo assignments until conflict constraint is unit
```

The major difference between CDCL and DPLL lies in the lookback techniques utilized to recover from conflicts: CDCL applies an *analyze* step that strengthens the input by adding a conflict constraint, and it also performs a *backjump* to a point where the conflict constraint is unit (yields some deterministic consequence by propagation). That is, CDCL replaces systematic backtracking by backjumping, and flips of former decision literals by inferences due to conflict constraints.

Both the traditional (DPLL) and the modern (CDCL) approach to SAT solving have firm proof-theoretic foundations in propositional resolution [18, 19, 197]. While DPLL amounts to a restricted form of resolution, called tree-like, CDCL (with restarts, deliberately discarding an assignment at hand in order to start from scratch) has been shown to be polynomially equivalent to strictly more powerful general resolution.

1.1 Contributions of This Thesis

In contrast to SAT, ASP lacks formal frameworks for describing inferences conducted by solvers, which has led to large heterogeneity in the description of algorithms for ASP solving, ranging over solver-specific operational [3, 32, 64, 154, 209] and procedural [122, 123, 167, 168, 218] characterizations. This complicates identifying fundamental properties of algorithms, such as soundness and completeness, as well as formal comparisons between them. Hence, one part of our work is motivated by the desire to converge the various heterogeneous characterizations of current ASP solvers' inferences by developing common proof-theoretic foundations. The proof-theoretic perspective allows us to state results in a general, rather than a solver-specific, way and to study inferences by their admissibility, rather than from an implementation point of view.

Despite the close relationship between ASP and SAT, state-of-the-art lookback techniques exploited in CDCL, such as backjumping, conflict-driven learning, and restarts, were not yet established in native ASP solvers. In fact, previous approaches to adopt such techniques [168, 200, 218] are rather implementation-specific and lack generality. Hence, our second goal is to devise modern ASP solving procedures that facilitate the integration of advanced lookback techniques. To this end, we show that all inferences in ASP

can be reduced to unit propagation and provide a self-contained algorithmic framework for conflict-driven ASP solving. Beyond the basic decision problem of answer set existence, we address enumeration in two settings: enumerating entire answer sets and their projections to a subvocabulary, respectively. In both settings, we aim at non-intrusive extensions of a decision procedure that enable repetition-free enumeration in polynomial space. Our approach to conflict-driven ASP solving is implemented in *clasp* [95, 97, 99], an award-winning[3] Boolean constraint solver with ASP as its core area.

In summary, the main contributions of this thesis are as follows:

1. We provide tableau frameworks for the construction of answer sets of normal logic programs, show that they allow for simulating a variety of existing ASP solving approaches, and establish exponential separations w.r.t. different concepts of case analyses.

2. We extend our tableau methods to logic programs with aggregates [67, 209], characterize inferences on cardinality constraints and disjunctions, and demonstrate the soundness and completeness of our approach w.r.t. an established general answer set semantics [69].

3. We characterize the semantics of normal logic programs in terms of Boolean constraints, develop an algorithmic framework for ASP solving by means of advanced lookback techniques, and elaborate upon its formal properties.

4. We provide an extension of CDCL-style decision procedures to repetition-free solution enumeration in polynomial space, develop an algorithm maintaining these properties in the enumeration of projections of solutions, and demonstrate the soundness and completeness of our enumeration methods.

The work on tableau methods (first two items above) has been conducted jointly with Torsten Schaub. Constraint-based semantic characterizations and algorithms (last two items above) have been devised jointly with Benjamin Kaufmann and Torsten Schaub, and André Neumann provided valuable support in algorithms' implementation and empirical evaluation. Theoretical foundations and formal elaboration of the contributions of this thesis are original work by the author.

Part of the results presented in thesis have been published (or are accepted for publication) in [4, 94, 95, 98, 99, 111, 112, 113],[4] coauthored by the author of this thesis.

1.1.1 Further Contributions

The author has contributed to research in declarative programming, mainly in the field of ASP, also beyond what can be presented in this thesis. Investigated subjects include:

- semantic foundations of ASP and equivalence notions for logic programs [101, 102, 103, 109, 116], jointly with Joohyung Lee, Yuliya Lierler, Torsten Schaub, Hans Tompits, and Stefan Woltran,

[3] Amongst others, *clasp* successfully participated in the 2009 ASP competition [47] (`http://dtai.cs.kuleuven.be/events/ASP-competition`), the 2009 Pseudo-Boolean competition (`http://www.cril.univ-artois.fr/PB09`), and the 2009 SAT competition (`http://www.satcompetition.org`).

[4] [111] received the Best Paper Award of the 22nd International Conference on Logic Programming (ICLP'06).

- declarative debugging methodologies for logic programs [27, 28, 106], jointly with Martin Brain, Jörg Pührer, Torsten Schaub, Hans Tompits, and Stefan Woltran,
- ground instantiation of logic programs [82, 90, 91, 114], jointly with Roland Kaminski, Benjamin Kaufmann, Arne König, Max Ostrowski, Torsten Schaub, and Sven Thiele,
- advanced features of *clasp* [53, 84, 93, 96, 97], jointly with Christian Drescher, Roland Kaminski, Benjamin Kaufmann, André Neumann, and Torsten Schaub,
- parallelization of *clasp* [59, 88, 207], jointly with Enrico Ellguth, Markus Gusowski, Roland Kaminski, Benjamin Kaufmann, Stefan Liske, Torsten Schaub, Lars Schneidenbach, and Bettina Schnor,
- extensions of *clasp* to disjunctive programs [52] and finite-domain variables [105], jointly with Christian Drescher, Torsten Grote, Benjamin Kaufmann, Arne König, Max Ostrowski, and Torsten Schaub,
- incremental ASP solving [83] along with applications to stream reasoning [78], action languages [79], automated planning [89], and finite model computation [107, 108],[5] jointly with Torsten Grote, Roland Kaminski, Benjamin Kaufmann, Murat Knecht, Max Ostrowski, Orkunt Sabuncu, Torsten Schaub, and Sven Thiele,
- multi-criteria optimization methods [85] along with their application to Linux package configuration [86] and meta-programming techniques implementing complex optimization criteria in ASP [92], jointly with Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub,
- application of ASP to systems biology [80, 100, 115], jointly with Carito Guziolowski, Mihail Ivanchev, Arne König, Torsten Schaub, Anne Siegel, Sven Thiele, and Philippe Veber,
- application of ASP to multiprocessor system on chip synthesis [134, 135], jointly with Christophe Bobda, Harold Ishebabi, Philipp Mahr, and Torsten Schaub, and
- assistance in ASP competitions in 2007 [104] and 2009 [47], jointly with Stephen Bond, Marc Denecker, Lengning Liu, Gayathri Namasivayam, André Neumann, Torsten Schaub, Mirosław Truszczyński, Joost Vennekens, and many more contributors as well as participants.

1.2 Organization of This Thesis

The main part of this thesis is organized as follows.

Chapter 2 introduces the syntax and semantics of (propositional) normal logic programs, Boolean assignments along with constraints expressed by nogoods, and an adaption of the traditional concept of an unfounded set [159, 216] to our setting.

[5] [107] was selected as one of the two papers of the 12th European Conference on Logics in Artificial Intelligence (JELIA'10) to be delegated to the "Large Track of Best Papers from Sister Conferences" of the 22nd International Joint Conference on Artificial Intelligence (IJCAI'11).

In Chapter 3, we provide tableau frameworks for the construction of answer sets of logic programs. We begin with inferences specialized to normal programs and show that existing logic programming concepts as well as ASP solving approaches can be characterized within our framework. We then generalize our methodology and apply it to logic programs with aggregates: in particular, we show how cardinality constraints and disjunctions can be accommodated in our approach. Moreover, we investigate the proof complexity of tableau methods and establish exponential separations w.r.t. different concepts of case analyses. Finally, we discuss related work as well as the achieved results.

In Chapter 4, we characterize answer sets of normal programs in terms of their induced constraints. These constraints provide the basis for developing a self-contained algorithmic framework for conflict-driven ASP solving, incorporating advanced look-back techniques. After considering the basic decision problem of answer set existence, we devise novel backtracking schemes allowing for the repetition-free enumeration of answer sets and their projections to a subvocabulary, respectively, in polynomial space. Importantly, our enumeration algorithms are designed to be non-intrusive in the sense that they do not interfere with the underlying decision procedure a priori, but only after solutions have been obtained. The described algorithms are implemented in *clasp*, and we present some experimental results demonstrating their effectiveness. Finally, we discuss related work as well as the achieved results.

Chapter 5 concludes the main part of this thesis. Unabridged traces of the enumeration algorithms presented in Section 4.4 are provided in Appendix A. Proofs of formal results are given in Appendix B.

Chapter 2

Background

This chapter introduces (propositional) normal logic programs under answer set semantics, Boolean assignments and nogoods, and unfounded sets. The answer sets of logic programs can be viewed as Boolean assignments satisfying constraints induced by a program, and the concept of an unfounded set [216, 159] is useful to identify such constraints. Unfounded sets are linked to loops [167, 156], as they provide a syntactic characterization of "interesting" unfounded sets.

The outline of this chapter is as follows. In Section 2.1, we introduce the formal syntax and semantics of normal logic programs. Section 2.2 defines Boolean assignments and nogoods, which provide us with a canonical framework to represent (Boolean) constraints and solutions for them. In Section 2.3, we adapt the concept of an unfounded set to the setting of Boolean assignments, and we identify fundamental properties that are exploited in later chapters.

The background presented in Section 2.1 and 2.2 can be viewed as common knowledge. The unfounded set notion developed in Section 2.3 and its properties have in parts been presented in [4, 99, 113], coauthored by the author of this thesis.

2.1 Normal Logic Programs

Given a denumerable alphabet $\mathcal{P}$, a (*propositional*) *normal* (*logic*) *program* is a finite set of rules of the form

$$p_0 \leftarrow p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n \tag{2.1}$$

where $0 \leq m \leq n$ and each $p_i \in \mathcal{P}$ is an *atom* for $0 \leq i \leq n$. A *literal* is an atom p or its (default) negation $not\ p$. For a rule r as in (2.1), we define the following notations:

$$\begin{array}{rcl} head(r) & = & p_0 \\ body(r) & = & \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\} \end{array} \tag{2.2}$$

Atom $head(r)$ is the *head* of r, and the set $body(r)$ of literals is the *body* of r. We sometimes write a rule r as $head(r) \leftarrow body(r)$. For a set B of literals, let $B^+ = B \cap \mathcal{P}$ and $B^- = \{p \in \mathcal{P} \mid not\ p \in B\}$. Accordingly, for $body(r)$ as in (2.2), we get:

$$\begin{array}{rcl} body(r)^+ & = & \{p_1, \ldots, p_m\} \\ body(r)^- & = & \{p_{m+1}, \ldots, p_n\} \end{array}$$

For a normal program Π and an atom $p \in \mathcal{P}$, we define the following notations:

$$\begin{array}{rcl} atom(\Pi) & = & \bigcup_{r \in \Pi} \left(\{head(r)\} \cup body(r)^+ \cup body(r)^-\right) \\ body(\Pi) & = & \{body(r) \mid r \in \Pi\} \\ body_\Pi(p) & = & \{body(r) \mid r \in \Pi, head(r) = p\} \end{array}$$

That is, $atom(\Pi)$ denotes the set of atoms occurring in Π, and $body(\Pi)$ denotes the set of bodies occurring in Π. Furthermore, $body_\Pi(p)$ qualifies the set of all bodies B such that a rule $p \leftarrow B$ belongs to Π.

The semantics of a normal program Π is given by the answer sets of Π, which can be viewed as classical models of Π (identifying rules with implications, rule bodies with conjunctions, and not with classical negation) such that all entailed atoms are necessarily derived by the rules in Π. In the following, we represent an interpretation by the set of its entailed atoms. Then, a set X of atoms is a *model* of Π if $head(r) \in X$, $body(r)^+ \not\subseteq X$, or $body(r)^- \cap X \neq \emptyset$ holds for every rule $r \in \Pi$. The definition of an answer set, according to [119], builds on the concept of a reduct and its least model. The *reduct*, denoted by Π^X, of Π w.r.t. a set X of atoms is the following program:

$$\Pi^X = \{head(r) \leftarrow body(r)^+ \mid r \in \Pi, body(r)^- \cap X = \emptyset\}$$

Essentially, Π^X is obtained from Π by dropping all rules whose negative body parts are not satisfied w.r.t. X and by removing the negative literals from bodies of the remaining rules. Note that Π^X consists of definite clauses, so that there is a unique least model of Π^X, denoted by $Cn(\Pi^X)$. Given this, X is an *answer set* of Π if $Cn(\Pi^X) = X$. Beyond this "traditional" characterization of answer sets, there are plenty alternative ones [162]. We introduce three of them in Section 2.3, 3.2, and 3.3.

Example 2.1. *Consider the following normal program:*

$$\Pi_1 = \left\{ \begin{array}{ll} r_1: & a \leftarrow \\ r_2: & c \leftarrow not\ b, not\ d \\ r_3: & d \leftarrow a, not\ c \end{array} \right\}$$

In view of $body(r_1)^- = \emptyset$*, we have that* $r_1 \in \Pi_1^X$ *and* $a \in Cn(\Pi_1^X)$ *for any set* X *of atoms; furthermore,* $body_{\Pi_1}(b) = \emptyset$ *implies that* $b \notin Cn(\Pi_1^X)$*. Hence, for any answer set* X *of* Π_1*, it holds that* $\{a\} \subseteq X \subseteq \{a, c, d\}$*. We obtain the following reducts w.r.t. the four candidate sets:*

$$\Pi_1^{\{a\}} = \left\{ \begin{array}{l} a \leftarrow \\ c \leftarrow \\ d \leftarrow a \end{array} \right\}$$

$$\Pi_1^{\{a,c\}} = \left\{ \begin{array}{l} a \leftarrow \\ c \leftarrow \end{array} \right\}$$

$$\Pi_1^{\{a,d\}} = \left\{ \begin{array}{l} a \leftarrow \\ d \leftarrow a \end{array} \right\}$$

$$\Pi_1^{\{a,c,d\}} = \left\{ \begin{array}{l} a \leftarrow \end{array} \right\}$$

We have that $Cn(\Pi_1^{\{a\}}) = \{a, c, d\} \neq \{a\}$*,* $Cn(\Pi_1^{\{a,c\}}) = \{a, c\}$*,* $Cn(\Pi_1^{\{a,d\}}) = \{a, d\}$*, and* $Cn(\Pi_1^{\{a,c,d\}}) = \{a\} \neq \{a, c, d\}$*. That is,* $\{a, c\}$ *and* $\{a, d\}$ *are the two answer sets of* Π_1*.*

2.2 Boolean Assignments and Nogoods

A (*Boolean*) *assignment* $\mathbf{A}$ over a *domain*, $dom(\mathbf{A})$, is a sequence $(\sigma_1, \ldots, \sigma_n)$ of *entries* σ_i of the form $\boldsymbol{T}v_i$ or $\boldsymbol{F}v_i$, where $v_i \in dom(\mathbf{A})$ for $1 \leq i \leq n$. An entry $\boldsymbol{T}v$ or $\boldsymbol{F}v$ expresses that v is *true* or *false*, respectively. We refer to the *variable* in an entry σ by $var(\sigma) = v$ and denote the *complement* of σ by $\overline{\sigma}$, that is, $\overline{\boldsymbol{T}v} = \boldsymbol{F}v$ and $\overline{\boldsymbol{F}v} = \boldsymbol{T}v$. We sometimes abuse notation and identify an assignment with the set of its contained entries. Given this, we access the true and the false variables in $\mathbf{A}$ via $\mathbf{A}^{\boldsymbol{T}} = \{v \in dom(\mathbf{A}) \mid \boldsymbol{T}v \in \mathbf{A}\}$ and $\mathbf{A}^{\boldsymbol{F}} = \{v \in dom(\mathbf{A}) \mid \boldsymbol{F}v \in \mathbf{A}\}$. We say that $\mathbf{A}$ is *contradictory* if $\mathbf{A}^{\boldsymbol{T}} \cap \mathbf{A}^{\boldsymbol{F}} \neq \emptyset$; otherwise, $\mathbf{A}$ is *non-contradictory*. Furthermore, $\mathbf{A}$ is *total* if it is non-contradictory and $\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}} = dom(\mathbf{A})$. In order to accommodate our proof-theoretic and constraint-based characterizations of answer sets in Chapter 3 and 4, respectively, for a normal program Π, we fix $dom(\mathbf{A})$ to $atom(\Pi) \cup body(\Pi)$ in the sequel. For instance, with Π_1 from Example 2.1, the (non-total) assignments $(\boldsymbol{T}\emptyset, \boldsymbol{F}b)$ and $(\boldsymbol{F}b, \boldsymbol{T}\emptyset)$ map the empty body of rule r_1 to true and the atom b to false, and the assignment $(\boldsymbol{F}b, \boldsymbol{T}\emptyset, \boldsymbol{T}a, \boldsymbol{T}c, \boldsymbol{F}\{a, not\ c\}, \boldsymbol{F}d, \boldsymbol{T}\{not\ b, not\ d\})$ is total.

For representing constraints, we take advantage of the concept of a nogood (cf. [45, 201]). In our setting, a *nogood* is a set $\{\sigma_1, \ldots, \sigma_m\}$ of entries, expressing that any assignment containing $\sigma_1, \ldots, \sigma_m$ is unintended. Accordingly, a total assignment $\mathbf{A}$ is a *solution* for a set Δ of nogoods if $\delta \not\subseteq \mathbf{A}$ for all $\delta \in \Delta$. For instance, given the domain $atom(\Pi_1) \cup body(\Pi_1) = \{a, b, c, d, \emptyset, \{not\ b, not\ d\}, \{a, not\ c\}\}$, the total assignment $(\boldsymbol{F}b, \boldsymbol{T}\emptyset, \boldsymbol{T}a, \boldsymbol{T}c, \boldsymbol{F}\{a, not\ c\}, \boldsymbol{F}d, \boldsymbol{T}\{not\ b, not\ d\})$ is a solution for the following set of nogoods:

$$\Delta = \left\{ \begin{array}{l} \{\boldsymbol{T}b\}, \{\boldsymbol{F}\emptyset\}, \{\boldsymbol{F}a, \boldsymbol{T}\emptyset\}, \{\boldsymbol{T}a, \boldsymbol{F}\emptyset\}, \\ \{\boldsymbol{F}c, \boldsymbol{T}\{not\ b, not\ d\}\}, \{\boldsymbol{T}c, \boldsymbol{F}\{not\ b, not\ d\}\}, \\ \{\boldsymbol{F}\{not\ b, not\ d\}, \boldsymbol{F}b, \boldsymbol{F}d\}, \\ \{\boldsymbol{T}\{not\ b, not\ d\}, \boldsymbol{T}b\}, \{\boldsymbol{T}\{not\ b, not\ d\}, \boldsymbol{T}d\}, \\ \{\boldsymbol{F}d, \boldsymbol{T}\{a, not\ c\}\}, \{\boldsymbol{T}d, \boldsymbol{F}\{a, not\ c\}\}, \\ \{\boldsymbol{F}\{a, not\ c\}, \boldsymbol{T}a, \boldsymbol{F}c\}, \{\boldsymbol{T}\{a, not\ c\}, \boldsymbol{F}a\}, \{\boldsymbol{T}\{a, not\ c\}, \boldsymbol{T}c\} \end{array} \right\}$$

Since we deal with propositional logic programs, their answer sets can be characterized in terms of solutions for suitable sets of nogoods. Such sets of nogoods are developed in Section 4.1.

2.3 Unfounded Sets

Unfounded sets characterize atoms that cannot be derived from a logic program, so that they do not belong to the least model of its reduct. We below reformulate the "traditional" unfounded set definition by Van Gelder, Ross, and Schlipf [216] on the basis of assignments, then simplify it, and finally provide fundamental properties of our simplified unfounded set notion.

To begin with, for a normal program Π and a set $U \subseteq atom(\Pi)$, we define the *external bodies* of U for Π, denoted by $EB_\Pi(U)$, as follows:

$$EB_\Pi(U) \quad = \quad \{body(r) \mid r \in \Pi, head(r) \in U, body(r)^+ \cap U = \emptyset\}$$

Observe that an external body belongs to a rule r whose head is in U, and it does not contain any element of U in its positive body part. Hence, if $head(r) \leftarrow body(r)^+$ belongs to Π^X for some set X of atoms, it may justify the inclusion of an atom in U, namely, of $head(r)$, in $Cn(\Pi^X)$. In turn, if all elements of $EB_\Pi(U)$ evaluate to false w.r.t. a model X of Π, it follows that $Cn(\Pi^X) \cap U = \emptyset$.

The impact of external bodies on the least model of a program's reduct motivates the following adaption of Definition 3.1 in [216] to our concept of an assignment.

Definition 2.1. *Let Π be a normal program, $\mathbf{A}$ an assignment, and $U \subseteq atom(\Pi)$.*

Then, we define U as a GRS-unfounded set of Π w.r.t. $\mathbf{A}$ if $EB_\Pi(U) \subseteq \{B \in body(\Pi) \mid (B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\}$.

Theorem 5.4 in [216] can then be adapted as follows.

Proposition 2.1. *Let Π be a normal program and $\mathbf{A}$ a non-contradictory assignment.*

If $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is a model of Π, then we have that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is an answer set of Π iff $U \cap \mathbf{A}^{\boldsymbol{T}} = \emptyset$ holds for every GRS-unfounded set U of Π w.r.t. $\mathbf{A}$.

Note that Definition 2.1 does not make use of bodies assigned by $\mathbf{A}$. By considering them, we can simplify GRS-unfounded sets in the following way.

Definition 2.2. *Let Π be a normal program, $\mathbf{A}$ an assignment, and $U \subseteq atom(\Pi)$.*

Then, we define U as an unfounded set of Π w.r.t. $\mathbf{A}$ if $EB_\Pi(U) \subseteq \mathbf{A}^{\boldsymbol{F}}$.

Unlike the GRS-unfounded set definition, which checks for some false literal in each external body of an unfounded set, our simplification requires the external bodies to be assigned to false.

Example 2.2. *Consider the following normal program:*

$$\Pi_2 \quad = \quad \left\{ \begin{array}{ll} r_1: & a \leftarrow not\ b \\ r_2: & b \leftarrow not\ a \\ r_3: & c \leftarrow a \\ r_4: & c \leftarrow b, d \\ r_5: & d \leftarrow b, c \\ r_6: & d \leftarrow e \\ r_7: & e \leftarrow b, not\ a \\ r_8: & e \leftarrow c, d \end{array} \right\}$$

For $U = \{d, e\}$, we get $EB_{\Pi_2}(U) = \{\{b, c\}, \{b, not\ a\}\}$. Hence, U is an unfounded set of Π_2 w.r.t. $\mathbf{A} = (\boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\})$, but U is not a GRS-unfounded set of Π_2 w.r.t. $\mathbf{A}$. On the other hand, we have that U is a GRS-unfounded set of Π_2 w.r.t. $\mathbf{B} = (\boldsymbol{F}b)$ because the positive body literal b in $\{b, c\}$ and $\{b, not\ a\}$ is false, but U is not an unfounded set of Π_2 w.r.t. $\mathbf{B}$.

The differences between GRS-unfounded sets and our simplified unfounded set notion are due to mismatches between assigned atoms and bodies. To enable a comparison, we thus define properties that restrict such mismatches.

Definition 2.3. *Let Π be a normal program and $\mathbf{A}$ an assignment.*

Then, we define $\mathbf{A}$ as

1. *body-saturated for Π if $\{B \in body(\Pi) \mid (B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\} \subseteq \mathbf{A}^{\boldsymbol{F}}$;*
2. *body-synchronized for Π if $\{B \in body(\Pi) \mid (B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\} = \mathbf{A}^{\boldsymbol{F}} \cap body(\Pi)$.*

In words, body-saturation requires that bodies containing false literals must likewise be assigned to false; if the converse holds as well, we have body-synchronization.

Based on these properties, we now formalize the relationships between GRS-unfounded sets and our unfounded set notion.

Proposition 2.2. *Let Π be a normal program, $\mathbf{A}$ an assignment, and $U \subseteq atom(\Pi)$.*

If $\mathbf{A}$ is body-saturated for Π, then we have that U is an unfounded set of Π w.r.t. $\mathbf{A}$ if U is a GRS-unfounded set of Π w.r.t. $\mathbf{A}$.

Proposition 2.3. *Let Π be a normal program, $\mathbf{A}$ an assignment, and $U \subseteq atom(\Pi)$.*

If $\mathbf{A}$ is body-synchronized for Π, then we have that U is an unfounded set of Π w.r.t. $\mathbf{A}$ iff U is a GRS-unfounded set of Π w.r.t. $\mathbf{A}$.

Proposition 2.2 shows that any GRS-unfounded set can be turned into an unfounded set by adding entries for bodies containing false literals to an assignment, in this way establishing body-saturation. The stronger condition of body-synchronization is required in Proposition 2.3 to guarantee that an unfounded set is GRS-unfounded as well.

The following example illustrates that, in general, it is not straightforward to make an assignment body-synchronized by adding entries.

Example 2.3. *Reconsider Π_2 and $U = \{d, e\}$ from Example 2.2, and recall that $EB_{\Pi_2}(U) = \{\{b, c\}, \{b, not\ a\}\}$. Also recall that U is GRS-unfounded for Π_2 w.r.t. $\mathbf{B} = (\boldsymbol{F}b)$, but not unfounded. The reason for this mismatch is that $\mathbf{B}$ is not body-saturated for Π_2. When we add entries for bodies that are false to $\mathbf{B}$, we obtain the up to the order of entries unique assignment $\mathbf{B}' = (\boldsymbol{F}b, \boldsymbol{F}\{b, d\}, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\})$. Since $EB_{\Pi_2}(U) \subseteq (\mathbf{B}')^{\boldsymbol{F}}$, we have that U is an unfounded set of Π_2 w.r.t. $\mathbf{B}'$. On the other hand, U is unfounded for Π_2 w.r.t. $\mathbf{A} = (\boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\})$, but not GRS-unfounded. For turning U into a GRS-unfounded set, we could extend $\mathbf{A}$ by $\boldsymbol{F}b$ or both $\boldsymbol{F}c$ and $\boldsymbol{T}a$. Hence, there are several (minimal) body-synchronized extensions of $\mathbf{A}$.*

In view of Proposition 2.3, we immediately derive the following from Proposition 2.1.

Corollary 2.4. *Let Π be a normal program and $\mathbf{A}$ a non-contradictory assignment.*

If $\mathbf{A}$ is body-synchronized for Π and if $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is a model of Π, then we have that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is an answer set of Π iff $U \cap \mathbf{A}^{\boldsymbol{T}} = \emptyset$ holds for every unfounded set U of Π w.r.t. $\mathbf{A}$.

Note that the (intended) total assignments characterized further in Chapter 3 and 4 are body-synchronized and correspond to models. Furthermore, if U is unfounded w.r.t. an assignment $\mathbf{A}$, it is clear from Definition 2.2 that U remains unfounded w.r.t. any total extension $\mathbf{A}'$ of $\mathbf{A}$. Hence, the requirement that $U \cap (\mathbf{A}')^{\boldsymbol{T}} = \emptyset$ tells us that all atoms of U must be false in $\mathbf{A}'$. That is, unfounded sets provide means to infer atoms that must necessarily be false.

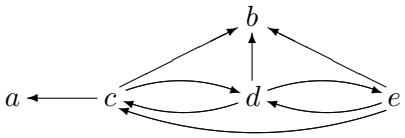
In principle, arbitrary unfounded sets can be used to identify necessarily false atoms. However, there are simple (structural) properties of logic programs that allow for focusing the consideration of unfounded sets to "interesting" ones. In the remainder of this section, we provide such properties and apply them to our unfounded set notion.

The (*positive*) *dependency graph* of a normal program Π, denoted by $DG(\Pi)$, is the following directed graph:

$$DG(\Pi) \quad = \quad \big(atom(\Pi), \{(head(r), p) \mid r \in \Pi, p \in body(r)^+\}\big)$$

Such a graph allows us to identify circular positive dependencies between atoms. According to [167], a non-empty $L \subseteq atom(\Pi)$ is a *loop* of Π if, for every pair $p \in L, q \in L$ (including $p = q$), there is a path of non-zero length from p to q in $DG(\Pi)$ such that all vertices in the path belong to L. We denote the set of all loops of Π by $loop(\Pi)$. The main interest of loops is that they can be unfounded even if each contained atom is (circularly) supported by some rule with non-false body.

Example 2.4. *The dependency graph $DG(\Pi_2)$ of Π_2 from Example 2.2 looks as follows:*



Observe that $\{c, d\}$, $\{d, e\}$, and $\{c, d, e\}$ are all non-empty sets of atoms such that their elements reach one another via (loop-internal) paths of non-zero length. Hence, we have that $loop(\Pi_2) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$.

We now begin with our consideration of structural properties of unfounded sets. Under the assumption of body-saturation, we may eliminate false atoms from an unfounded set in order to obtain an unfounded set of non-false atoms only.

Proposition 2.5. *Let Π be a normal program, $\mathbf{A}$ an assignment, and U an unfounded set of Π w.r.t. $\mathbf{A}$.*

If $\mathbf{A}$ is body-saturated for Π, then we have that $U \setminus \mathbf{A}^{\boldsymbol{F}}$ is an unfounded set of Π w.r.t. $\mathbf{A}$.

Example 2.5. *For instance, $U = \{b, d, e\}$ is an unfounded set of Π_2 from Example 2.2 w.r.t. the body-saturated assignment $\mathbf{A} = (\boldsymbol{F}\{not\ a\}, \boldsymbol{F}b, \boldsymbol{F}\{b, d\}, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\})$. Proposition 2.5 tells us that $U \setminus \mathbf{A}^{\boldsymbol{F}} = \{d, e\}$ remains unfounded for Π_2 w.r.t. $\mathbf{A}$. That is, we may limit the attention to unfounded sets containing exclusively non-false atoms.*

The following is an immediate consequence of Proposition 2.5.

Corollary 2.6. *Let Π be a normal program, $\mathbf{A}$ an assignment, and U an unfounded set of Π w.r.t. $\mathbf{A}$.*

If $\mathbf{A}$ is body-saturated for Π and if $U \not\subseteq \mathbf{A}^{\boldsymbol{F}}$, then we have that $U \setminus \mathbf{A}^{\boldsymbol{F}}$ is a non-empty unfounded set of Π w.r.t. $\mathbf{A}$.

In what follows, we exploit loops to confine the consideration of unfounded sets. To accomplish this, we introduce atom-saturation as a property dual to body-saturation.

Definition 2.4. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment.*

Then, we define $\mathbf{A}$ *as atom-saturated for* Π *if* $\{p \in atom(\Pi) \mid body_\Pi(p) \subseteq \mathbf{A}^{\boldsymbol{F}}\} \subseteq \mathbf{A}^{\boldsymbol{F}}$.

In words, atom-saturation requires that atoms not supported by any rule with non-false body must be assigned to false.

Given an atom-saturated assignment, we have that every non-empty unfounded set of non-false atoms contains some unfounded loop.

Proposition 2.7. *Let* Π *be a normal program,* $\mathbf{A}$ *an assignment, and* $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ *a non-empty unfounded set of* Π *w.r.t.* $\mathbf{A}$.

If $\mathbf{A}$ *is atom-saturated for* Π*, then we have that there is some unfounded set* $L \subseteq U$ *of* Π *w.r.t.* $\mathbf{A}$ *such that* $L \in loop(\Pi)$.

Example 2.6. *For instance,* $U = \{d, e\}$ *is an unfounded loop of* Π_2 *from Example 2.2 w.r.t. the atom-saturated assignment* $\mathbf{A} = (\boldsymbol{F}\{not\ a\}, \boldsymbol{F}b, \boldsymbol{F}\{b, d\}, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\})$. *In contrast, the only non-empty unfounded set* $\{a\}$ *of* Π_2 *w.r.t. the assignment* $\mathbf{B} = (\boldsymbol{F}\{not\ b\})$ *is not a loop of* Π_2*, but* $\mathbf{B}$ *is not atom-saturated for* Π_2.

Since a logic program may yield exponentially many loops, which can be unfounded separately w.r.t. different assignments, it is impractical to identify (arbitrary) loops a priori. However, the *non-trivial* strongly connected components of a dependency graph limit the atoms that can jointly belong to (unfounded) loops.[1] The fact that the consideration of unfounded sets can be confined to non-trivial strongly connected components is an immediate consequence of Proposition 2.7.

Corollary 2.8. *Let* Π *be a normal program,* $\mathbf{A}$ *an assignment, and* $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ *a non-empty unfounded set of* Π *w.r.t.* $\mathbf{A}$.

If $\mathbf{A}$ *is atom-saturated for* Π*, then we have that there is some non-empty unfounded set* $U' \subseteq U$ *of* Π *w.r.t.* $\mathbf{A}$ *such that all* $p \in U'$ *belong to the same non-trivial strongly connected component of* $DG(\Pi)$.

By combining Corollary 2.6 and Proposition 2.7, we obtain the following fundament for unfounded set handling approaches described in Chapter 3 and 4.

Theorem 2.9. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment.*

If $\mathbf{A}$ *is both atom- and body-saturated for* Π *and if there is some unfounded set* U *of* Π *w.r.t.* $\mathbf{A}$ *such that* $U \not\subseteq \mathbf{A}^{\boldsymbol{F}}$*, then we have that there is some unfounded set* $L \subseteq U \setminus \mathbf{A}^{\boldsymbol{F}}$ *of* Π *w.r.t.* $\mathbf{A}$ *such that* $L \in loop(\Pi)$.

Regarding strongly connected components, we immediately derive the following from Theorem 2.9.

Corollary 2.10. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment.*

If $\mathbf{A}$ *is both atom- and body-saturated for* Π *and if there is some unfounded set* U *of* Π *w.r.t.* $\mathbf{A}$ *such that* $U \not\subseteq \mathbf{A}^{\boldsymbol{F}}$*, then we have that there is some non-empty unfounded set* $U' \subseteq U \setminus \mathbf{A}^{\boldsymbol{F}}$ *of* Π *w.r.t.* $\mathbf{A}$ *such that all* $p \in U'$ *belong to the same non-trivial strongly connected component of* $DG(\Pi)$.

[1] A strongly connected component of a directed graph is a maximal subgraph such that its vertices reach one another via paths. A strongly connected component is *non-trivial* if it contains some edge. Strongly connected components can be computed in linear time [212] and induce a partition of vertices.

Example 2.7. *Reconsider the dependency graph $DG(\Pi_2)$ shown in Example 2.4, and observe that the subgraph induced by $C = \{c, d, e\}$ is the only non-trivial strongly connected component. As already noted in Example 2.5, we have that $U = \{b, d, e\}$ is an unfounded set of Π_2 w.r.t. the atom- and body-saturated assignment $\mathbf{A} = (\boldsymbol{F}\{not\ a\}, \boldsymbol{F}b, \boldsymbol{F}\{b, d\}, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\})$. In view of Corollary 2.10, we conclude that some non-empty subset of $U \cap C = \{b, d, e\} \cap \{c, d, e\} = \{d, e\}$ is unfounded for Π_2 w.r.t. $\mathbf{A}$. Furthermore, Theorem 2.9 tells us that some unfounded subset of $\{d, e\}$ is a loop of Π_2. Along with $loop(\Pi_2) = \{\{c, d\}, \{d, e\}, \{c, d, e\}\}$ (cf. Example 2.4), it directly follows that $\{d, e\}$ is the unfounded loop contained in U. Finally, note that the set C of all atoms belonging to the non-trivial strongly connected component of $DG(\Pi_2)$ is not unfounded for Π_2 w.r.t. $\mathbf{A}$ because $\{a\} \in EB_{\Pi_2}(C) \setminus \mathbf{A}^{\boldsymbol{F}}$.*

The observation that a non-trivial strongly connected component may include an unfounded loop even if the set of all atoms in the component is not unfounded emphasizes that strongly connected components merely provide limits by which the consideration of unfounded sets can be confined. The unfounded sets within a component can, however, vary between assignments and must thus be considered individually.

Chapter 3

Tableaux for Answer Set Programming

This chapter develops proof-theoretic foundations for the construction of answer sets of logic programs. The basic idea is to generate a binary tree, called tableau [40], whose branches consist of a logic program along with a Boolean assignment. To this end, deterministic and non-deterministic tableau rules admit to extend a branch by appending entries, leading to a single or two extended branches, respectively. A tableau calculus is a collection of tableau rules, characterizing all tableaux that can be generated from an initial branch by applying the rules the calculus contains.

We mainly consider tableau calculi such that the leaves of their tableaux indicate successful or unsuccessful attempts to construct an answer set of a given logic program, and an entire tableau represents a traversal of the search space. Viewing that applications of deterministic and non-deterministic tableau rules amount to propagation and case analyses, respectively, tableau generation is closely related to computations with DPLL-style procedures. This relationship allows us to tightly characterize existing ASP solvers by associating particular tableau calculi with them, in the sense that computations of a solver correspond to tableaux of an associated calculus. Since tableau calculi characterize all tableaux constructible by applying their rules, they allow for analyzing the (best-case) complexities of computations with ASP solvers.

The outline of this chapter is as follows. In Section 3.1, we introduce tableau rules for constructing answer sets of normal programs. Section 3.2 relates particular tableau calculi to existing ASP solving approaches. In Section 3.3, we generalize our approach beyond the class of normal programs. The (best-case) complexities of tableaux constructible with particular calculi are analyzed in Section 3.4. Section 3.5 and 3.6 conclude the chapter by surveying related work and discussing the achieved results, respectively.

Parts of this chapter have also been presented in [111, 112, 113], coauthored by the author of this thesis.

3.1 Tableaux for Normal Logic Programs

We describe calculi consisting of tableau rules for the construction of answer sets of logic programs. A *tableau* for a logic program Π and an initial assignment $\mathbf{A}$ is a binary tree with the rules of Π and the entries of $\mathbf{A}$ at its root.[1] Further entries can be generated by applying tableau rules in the following standard way [40]: given a tableau rule and a branch in a tableau such that the prerequisites of the rule hold in the branch, the tableau can be extended by appending entries to the end of the branch as specified by the rule. Note that every *branch* corresponds to a pair $(\Pi, \mathbf{A})$; we draw on this relationship for identifying branches in the sequel. Moreover, as the tableau rules we present do not take the order of entries into account, we write assignments $\mathbf{A}$ as sets of entries, rather than sequences, throughout this chapter.

For some $v \in dom(\mathbf{A})$, we say that $\boldsymbol{T}v$ or $\boldsymbol{F}v$ can be deduced by a set $\mathcal{T}$ of tableau rules in a branch $(\Pi, \mathbf{A})$ if the entry can be generated by applying some rule in $\mathcal{T}$ other than *Cut* (see below). We let $D_{\mathcal{T}}(\Pi, \mathbf{A})$ denote the set of entries deducible by $\mathcal{T}$ in $(\Pi, \mathbf{A})$; $D^*_{\mathcal{T}}(\Pi, \mathbf{A})$ represents the set of entries in a smallest branch that extends $(\Pi, \mathbf{A})$ and is closed under $\mathcal{T}$, that is, $D_{\mathcal{T}}(\Pi, D^*_{\mathcal{T}}(\Pi, \mathbf{A})) \subseteq D^*_{\mathcal{T}}(\Pi, \mathbf{A})$. A branch $(\Pi, \mathbf{A})$ is *contradictory* if $\mathbf{A}$ is contradictory, and *non-contradictory* otherwise; $(\Pi, \mathbf{A})$ is *complete* (w.r.t. a tableau calculus $\mathcal{T}$) if it is contradictory or if $\mathbf{A}$ is total and $D_{\mathcal{T}}(\Pi, \mathbf{A}) \subseteq \mathbf{A}$. A tableau is *complete* if all of its branches are complete. A complete tableau for a logic program and the empty assignment such that all branches are contradictory is called a *refutation* for the program (meaning that the program has no answer set).

Our tableau rules for normal programs Π are shown in Figure 3.1. For convenience, they make use of two conjugation functions, $\boldsymbol{t}$ and $\boldsymbol{f}$. For a literal l, define:

$$\boldsymbol{t}l = \begin{cases} \boldsymbol{T}l & \text{if } l \in dom(\mathbf{A}) \\ \boldsymbol{F}v & \text{if } l = not\ v \text{ for } v \in dom(\mathbf{A}) \end{cases}$$

$$\boldsymbol{f}l = \begin{cases} \boldsymbol{F}l & \text{if } l \in dom(\mathbf{A}) \\ \boldsymbol{T}v & \text{if } l = not\ v \text{ for } v \in dom(\mathbf{A}) \end{cases}$$

In view of this, the *FTB* rule in *(a)* expresses that truth of a rule body can be deduced if the body's literals hold in a branch. Conversely, if the body is already assigned to false and all but one literal hold, the remaining literal must necessarily be false; this contrapositive argument is formalized by the *BFB* rule in *(b)*. Likewise, the tableau rules *FTA* and *FFB* in *(c)* and *(e)* capture straightforward conditions under which an atom must be assigned to true and a body to false, respectively. Their contrapositives are given by *BFA* and *BTB* in *(d)* and *(f)*. The remaining tableau rules in *(g)–(k)* are subject to provisos. For an application of *FFA* in *(g)*, deducing an unsupported atom p to be false, $(\S)$ stipulates that $B_1, \ldots, B_m$ comprise all bodies of rules with head p. Its contrapositive, the *BTA* rule in *(h)*, is also guided by $(\S)$. The outer structure of *WFN*$[\Omega]$ and *WFJ*$[\Omega]$ in *(i)* and *(j)*, aiming at unfounded sets, is similar to *FFA* and *BTA*, yet their proviso $(\dagger[\Omega])$ requires a concerned atom p to belong to some set $U \in \Omega$ such that $B_1, \ldots, B_m$ comprise all external bodies of U for Π. We below investigate two alternative options for Ω: $\Omega = 2^{atom(\Pi)}$ and $\Omega = loop(\Pi)$. Finally, $(\sharp[\Gamma])$ guides applications of the *Cut*$[\Gamma]$ rule in *(k)* by restricting cut objects v to members of Γ. For a normal program Π, we below consider different sets $\Gamma \subseteq atom(\Pi) \cup body(\Pi)$.[2] Note that a *Cut* application adds

[1] We do not mark the immanent validity of rules in Π by $\boldsymbol{T}$, as rules are not assigned by $\mathbf{A}$.

[2] The *Cut* rule may, in principle, introduce more general entries; this would however necessitate additional decomposition rules, leading to extended tableau calculi.

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n}{\boldsymbol{T}\{l_1, \ldots, l_n\}}$$

(a) Forward True Body (FTB)

$$\frac{\boldsymbol{F}\{l_1, \ldots, l_{i-1}, l_i, l_{i+1}, \ldots, l_n\} \quad \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_{i-1}, \boldsymbol{t}l_{i+1}, \ldots, \boldsymbol{t}l_n}{\boldsymbol{f}l_i}$$

(b) Backward False Body (BFB)

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \boldsymbol{T}\{l_1, \ldots, l_n\}}{\boldsymbol{T}p}$$

(c) Forward True Atom (FTA)

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \boldsymbol{F}p}{\boldsymbol{F}\{l_1, \ldots, l_n\}}$$

(d) Backward False Atom (BFA)

$$\frac{p \leftarrow l_1, \ldots, l_i, \ldots, l_n \quad \boldsymbol{f}l_i}{\boldsymbol{F}\{l_1, \ldots, l_i, \ldots, l_n\}}$$

(e) Forward False Body (FFB)

$$\frac{\boldsymbol{T}\{l_1, \ldots, l_i, \ldots, l_n\}}{\boldsymbol{t}l_i}$$

(f) Backward True Body (BTB)

$$\frac{\boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_m}{\boldsymbol{F}p}\ (\S)$$

(g) Forward False Atom (FFA)

$$\frac{\boldsymbol{T}p \quad \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_{i-1}, \boldsymbol{F}B_{i+1}, \ldots, \boldsymbol{F}B_m}{\boldsymbol{T}B_i}\ (\S)$$

(h) Backward True Atom (BTA)

$$\frac{\boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_m}{\boldsymbol{F}p}\ (\dagger[\Omega])$$

(i) Well-Founded Negation (WFN[Ω])

$$\frac{\boldsymbol{T}p \quad \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_{i-1}, \boldsymbol{F}B_{i+1}, \ldots, \boldsymbol{F}B_m}{\boldsymbol{T}B_i}\ (\dagger[\Omega])$$

(j) Well-Founded Justification (WFJ[Ω])

$$\frac{}{\boldsymbol{T}v \mid \boldsymbol{F}v}\ (\sharp[\Gamma])$$

(k) Cut (Cut[Γ])

$$\begin{array}{rcl} (\S) & : & p \in atom(\Pi), body_\Pi(p) \subseteq \{B_1, \ldots, B_m\} \subseteq body(\Pi) \\ (\dagger[\Omega]) & : & p \in U, U \in \Omega, EB_\Pi(U) \subseteq \{B_1, \ldots, B_m\} \subseteq body(\Pi) \\ (\sharp[\Gamma]) & : & v \in \Gamma \end{array}$$

Figure 3.1: Tableau rules for normal programs.

$$\left\{\begin{array}{ll} r_1: & a \leftarrow \\ r_2: & c \leftarrow not\ b, not\ d \\ r_3: & d \leftarrow a, not\ c \end{array}\right\}$$

$\boldsymbol{T}\emptyset$ (*FTB*)
$\boldsymbol{T}a$ (*FTA*)
$\boldsymbol{F}b$ (*FFA*)

$\boldsymbol{T}c$			$\boldsymbol{F}c$	(*Cut*$[atom(\Pi_1)]$)
$\boldsymbol{T}\{not\ b, not\ d\}$	(*BTA*)		$\boldsymbol{F}\{not\ b, not\ d\}$	(*BFA*)
$\boldsymbol{F}d$	(*BTB*)		$\boldsymbol{T}d$	(*BFB*)
$\boldsymbol{F}\{a, not\ c\}$	(*FFB*)		$\boldsymbol{T}\{a, not\ c\}$	(*FTB*)

Figure 3.2: Complete tableau of $\mathcal{T}_{\textit{smodels}}$ for Π_1 from Example 2.1 and the empty assignment.

entries $\boldsymbol{T}v$ and $\boldsymbol{F}v$ as the left and the right child to the end of a branch, thus reflecting nondeterminism in assigning v. With every other tableau rule, its consequent is appended to a branch, i.e., applications are deterministic.

Note that all deterministic tableau rules in Figure 3.1 are answer set preserving; this also applies to the *Cut* rule when considering both resulting branches. Different tableau calculi, viz., particular rule sets, yield characteristic correspondences. For instance, $D^*_{\{\textit{FTB,FTA,FFB,FFA}\}}(\Pi, \mathbf{A})$ corresponds to Fitting's operator [74], and $D^*_{\{\textit{FTB,FTA,FFB,WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A})$ amounts to the well-founded operator [216]. Similarly, we show below that $D^*_{\{\textit{(a)–(h)}\}}(\Pi, \mathbf{A})$ coincides with unit propagation on a program's completion [37, 6], and $D^*_{\{\textit{(a)–(h),WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A})$ captures *smodels*' propagation [209], that is, well-founded operator enhanced by backward propagation. Furthermore, the following tableau calculi are of particular interest:

$$\begin{aligned}
\mathcal{T}_{\textit{comp}} &= \{\textit{(a)–(h)}, \textit{Cut}[atom(\Pi) \cup \textit{body}(\Pi)]\} \\
\mathcal{T}_{\textit{smodels}} &= \{\textit{(a)–(h)}, \textit{WFN}[2^{atom(\Pi)}], \textit{Cut}[atom(\Pi)]\} \\
\mathcal{T}_{\textit{nomore}} &= \{\textit{(a)–(h)}, \textit{WFN}[2^{atom(\Pi)}], \textit{Cut}[\textit{body}(\Pi)]\} \\
\mathcal{T}_{\textit{nomore++}} &= \{\textit{(a)–(h)}, \textit{WFN}[2^{atom(\Pi)}], \textit{Cut}[atom(\Pi) \cup \textit{body}(\Pi)]\}
\end{aligned}$$

Example 3.1. *An exemplary complete tableau of $\mathcal{T}_{\textit{smodels}}$ is given in Figure 3.2, where rule applications are indicated by rule names, e.g., (Cut$[atom(\Pi_1)]$). Both branches in Figure 3.2 are non-contradictory. They comprise Π_1 along with total assignments: the left branch represents answer set $\{a, c\}$, while the right one gives answer set $\{a, d\}$.*

The tableau calculi that include *WFN*$[2^{atom(\Pi)}]$ to deal with unfounded sets are sound and complete, which can be formalized as follows.

Theorem 3.1. *Let Π be a normal program.*

Then, we have that the following holds for tableau calculi $\mathcal{T}_{\textit{smodels}}$, $\mathcal{T}_{\textit{nomore}}$, and $\mathcal{T}_{\textit{nomore++}}$:

1. *Every incomplete tableau for Π and $\emptyset$ can be extended to a complete tableau for Π and $\emptyset$.*

2. *Program* Π *has an answer set* X *iff every complete tableau for* Π *and* $\emptyset$ *has a unique non-contradictory branch* $(\Pi, \mathbf{A})$ *such that* $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$.

3. *Program* Π *has no answer set iff every complete tableau for* Π *and* $\emptyset$ *is a refutation.*

In particular, note that each of *Cut*$[atom(\Pi)]$, *Cut*$[body(\Pi)]$, and *Cut*$[atom(\Pi) \cup body(\Pi)]$ is sufficient to complete tableaux for Π and $\emptyset$. However, we show in Section 3.4 that different proof complexities are obtained w.r.t. such *Cut* variants. Moreover, as each of $\mathcal{T}_{smodels}$, $\mathcal{T}_{nomore}$, and $\mathcal{T}_{nomore++}$ admits a (unique) non-contradictory complete branch $(\Pi, \mathbf{A})$ in some tableau iff $(\Pi, \mathbf{A})$ belongs to every complete tableau for Π and $\emptyset$, Theorem 3.1 remains valid when replacing "every" by "some" in the second and the third item of its statement.

3.2 Characterizing Existing ASP Solvers

In this section, we discuss the relationships between the tableau rules in Figure 3.1 and existing ASP solving approaches. As it turns out, our tableau rules are well-suited for describing the main principles of a variety of ASP solvers. We however start in Section 3.2.1 by showing correspondences with familiar logic programming operators: Fitting's operator [74] and the well-founded operator [216]. Section 3.2.2 then covers traditional approaches to answer set computation for normal programs, including *smodels* [209], *dlv* [158], *nomore* [154], and *nomore++* [3]. Finally, we sketch in Section 3.2.3 how tableau rules relate to propagation principles of SAT-based solvers *assat* [167], *cmodels* [123], and *sag* [168] as well as to the approaches of native conflict-driven learning ASP solvers *smodels*$_{cc}$ [218] and *clasp* [95].

3.2.1 Fitting's Operator and Well-Founded Operator

Given a normal program Π and an assignment $\mathbf{A}$, the two operators in question can be defined in terms of the following sets of atoms:

$$\begin{aligned}
\mathbb{T}_\Pi(\mathbf{A}) &= \{head(r) \mid r \in \Pi, body(r)^+ \subseteq \mathbf{A}^{\boldsymbol{T}}, body(r)^- \subseteq \mathbf{A}^{\boldsymbol{F}}\} \\
\mathbb{N}_\Pi(\mathbf{A}) &= atom(\Pi) \setminus \{head(r) \mid r \in \Pi, \\
&\qquad\qquad (body(r)^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (body(r)^- \cap \mathbf{A}^{\boldsymbol{T}}) = \emptyset\} \\
\mathbb{U}_\Pi(\mathbf{A}) &= \bigcup_{U \subseteq atom(\Pi), EB_\Pi(U) \subseteq \{body(r) \mid r \in \Pi, (body(r)^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (body(r)^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\}} U
\end{aligned}$$

$\mathbb{T}_\Pi(\mathbf{A})$ contains the head atoms of rules whose bodies hold w.r.t. $\mathbf{A}$. In contrast, if an atom of $\mathbb{N}_\Pi(\mathbf{A})$ occurs in the head of any rule, then the body of the rule is falsified by $\mathbf{A}$. Given that neither $\mathbb{T}_\Pi(\mathbf{A})$ nor $\mathbb{N}_\Pi(\mathbf{A})$ make use of any entry over $body(\Pi)$, they can be viewed as functions mapping partial interpretations over atoms. In fact, for $\mathbf{A}' = \{\boldsymbol{T}p \mid p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)\} \cup \{\boldsymbol{F}p \mid p \in \mathbf{A}^{\boldsymbol{F}} \cap atom(\Pi)\}$, we have that $\mathbb{T}_\Pi(\mathbf{A}) = \mathbb{T}_\Pi(\mathbf{A}')$ and $\mathbb{N}_\Pi(\mathbf{A}) = \mathbb{N}_\Pi(\mathbf{A}')$. Similarly, it holds that $\mathbb{U}_\Pi(\mathbf{A}) = \mathbb{U}_\Pi(\mathbf{A}')$, where the idea is to reflect the union of all GRS-unfounded sets of Π w.r.t. $\mathbf{A}$ or $\mathbf{A}'$, respectively.[3] Given that, for every $p \in \mathbb{N}_\Pi(\mathbf{A})$, we have $EB_\Pi(\{p\}) \subseteq body_\Pi(p) \subseteq \{body(r) \mid r \in \Pi, (body(r)^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (body(r)^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\}$, it is always the case that $\mathbb{N}_\Pi(\mathbf{A}) \subseteq \mathbb{U}_\Pi(\mathbf{A})$, while the converse does not hold in general.

With the sets $\mathbb{T}_\Pi(\mathbf{A})$, $\mathbb{N}_\Pi(\mathbf{A})$, and $\mathbb{U}_\Pi(\mathbf{A})$ at hand, we can now make the partial interpretations obtained by Fitting's operator and the well-founded operator precise. The

[3] The union of all GRS-unfounded sets is also called "greatest unfounded set" [216].

following assignment amounts to the result of an application of Fitting's operator: $\{\boldsymbol{T}p \mid p \in \mathbb{T}_\Pi(\mathbf{A})\} \cup \{\boldsymbol{F}p \mid p \in \mathbb{N}_\Pi(\mathbf{A})\}$.[4] The result of an application of the well-founded operator is the assignment $\{\boldsymbol{T}p \mid p \in \mathbb{T}_\Pi(\mathbf{A})\} \cup \{\boldsymbol{F}p \mid p \in \mathbb{U}_\Pi(\mathbf{A})\}$. While both operators use $\mathbb{T}_\Pi(\mathbf{A})$ to infer true atoms, false atoms are obtained via either $\mathbb{N}_\Pi(\mathbf{A})$ or $\mathbb{U}_\Pi(\mathbf{A})$. Since $\mathbb{N}_\Pi(\mathbf{A}) \subseteq \mathbb{U}_\Pi(\mathbf{A})$, the result of Fitting's operator is always subsumed by the one of the well-founded operator.

Example 3.2. *Consider the following normal program:*

$$\Pi_3 \quad = \quad \left\{ \begin{array}{ll} r_1: & a \leftarrow not\ b \\ r_2: & b \leftarrow not\ a \\ r_3: & c \leftarrow b, not\ a \\ r_4: & c \leftarrow d \\ r_5: & d \leftarrow b, not\ a \\ r_6: & d \leftarrow c \\ r_7: & e \leftarrow c \\ r_8: & e \leftarrow d \end{array} \right\}$$

For $\mathbf{A} = \{\boldsymbol{T}a, \boldsymbol{F}b\}$*, we get* $\mathbb{T}_{\Pi_3}(\mathbf{A}) = \{a\}$*,* $\mathbb{N}_{\Pi_3}(\mathbf{A}) = \{b\}$*, and* $\mathbb{U}_{\Pi_3}(\mathbf{A}) = \{b, c, d, e\}$*. That is,* $\{\boldsymbol{T}a, \boldsymbol{F}b\}$ *is the result of applying Fitting's operator to* $\mathbf{A}$*, while* $\{\boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{F}c, \boldsymbol{F}d, \boldsymbol{F}e\}$ *is obtained with the well-founded operator.*

For linking the operators to tableau rules, the following result characterizes the sets of inferred atoms in terms of *FTB* plus *FTA*, and by *FFB* along with either *FFA* or *WFN*$[2^{atom(\Pi)}]$, respectively.

Proposition 3.2. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment.*
Then, we have that

1. $\mathbb{T}_\Pi(\mathbf{A}) = \big(D_{\{FTA\}}(\Pi, D_{\{FTB\}}(\Pi, \mathbf{A}))\big)^{\boldsymbol{T}}$;
2. $\mathbb{N}_\Pi(\mathbf{A}) = \big(D_{\{FFA\}}(\Pi, D_{\{FFB\}}(\Pi, \mathbf{A}))\big)^{\boldsymbol{F}}$;
3. $\mathbb{U}_\Pi(\mathbf{A}) = \Big(D_{\{WFN[2^{atom(\Pi)}]\}}(\Pi, D_{\{FFB\}}(\Pi, \mathbf{A}))\Big)^{\boldsymbol{F}}$.

On the right-hand sides, the application of either *FTB* or *FFB* serves as an intermediate step to propagate the truth of all or the falsity of some body literal to the body as such. The valuations of bodies are then exploited by *FTA*, *FFA*, and *WFN*$[2^{atom(\Pi)}]$, which in turn deduce atoms assigned to true and false, respectively, in an application of Fitting's operator or the well-founded operator.

Example 3.3. *For* Π_3 *from Example 3.2 and* $\mathbf{A} = \{\boldsymbol{T}a, \boldsymbol{F}b\}$*,* $D_{\{FTB\}}(\Pi_3, \mathbf{A}) = \{\boldsymbol{T}\{not\ b\}\}$ *yields* $D_{\{FTA\}}(\Pi_3, D_{\{FTB\}}(\Pi_3, \mathbf{A})) = \{\boldsymbol{T}a\}$*. Furthermore, we have that* $D_{\{FFB\}}(\Pi_3, \mathbf{A}) = \{\boldsymbol{F}\{not\ a\}, \boldsymbol{F}\{b, not\ a\}\}$*,* $D_{\{FFA\}}(\Pi_3, D_{\{FFB\}}(\Pi_3, \mathbf{A})) = \{\boldsymbol{F}b\}$*, and* $D_{\{WFN[2^{atom(\Pi_3)}]\}}(\Pi_3, D_{\{FFB\}}(\Pi_3, \mathbf{A})) = \{\boldsymbol{F}b, \boldsymbol{F}c, \boldsymbol{F}d, \boldsymbol{F}e\}$*. As stated in Proposition 3.2, these outcomes correspond to* $\mathbb{T}_{\Pi_3}(\mathbf{A}) = \{a\}$*,* $\mathbb{N}_{\Pi_3}(\mathbf{A}) = \{b\}$*, and* $\mathbb{U}_{\Pi_3}(\mathbf{A}) = \{b, c, d, e\}$.

[4] Recall that, throughout this chapter, we write assignments as sets of entries, rather than sequences.

$$\frac{p \leftarrow l_1, \ldots, l_n \quad \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n}{\boldsymbol{T}p}$$

Forward Inference (FI)

$$\frac{\boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_m}{\boldsymbol{F}p}\ (\lhd)$$

All Rules Canceled (ARC)

$$\frac{p' \leftarrow l'_1, \ldots, l'_i, \ldots, l'_n \quad \boldsymbol{T}p, \boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_m}{\boldsymbol{t}l'_i}\ (\rhd)$$

Contraposition for True Heads (CTH)

$$\frac{p \leftarrow l_1, \ldots, l_{i-1}, l_i, l_{i+1}, \ldots, l_n \quad \boldsymbol{F}p, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_{i-1}, \boldsymbol{t}l_{i+1}, \ldots, \boldsymbol{t}l_n}{\boldsymbol{f}l_i}$$

Contraposition for False Heads (CFH)

$$\frac{\boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_m}{\boldsymbol{F}p}\ (\Diamond)$$

At Most (AM)

$(\lhd)$: $p \in atom(\Pi), \{r \in \Pi \mid head(r) = p, body(r) \cap \{l_1, \ldots, l_m\} = \emptyset\} = \emptyset$
$(\rhd)$: $p \in atom(\Pi), \{r \in \Pi \mid head(r) = p, body(r) \cap \{l_1, \ldots, l_m\} = \emptyset\} \subseteq \{p' \leftarrow l'_1, \ldots, l'_i, \ldots, l'_n\}$
$(\Diamond)$: $p \in U, U \subseteq atom(\Pi), EB_\Pi(U) \subseteq \{body(r) \mid r \in \Pi, body(r) \cap \{l_1, \ldots, l_m\} \neq \emptyset\}$

Figure 3.3: Deterministic tableau rules for traditional (atom-based) ASP solvers.

Although we omit further details, we note that the correspondences established in Proposition 3.2 carry forward to iterated applications and fixpoints of Fitting's operator and the well-founded operator, respectively. In particular, the entries over $atom(\Pi)$ in $D^*_{\{FTB,FTA,FFB,FFA\}}(\Pi, \emptyset)$ and $D^*_{\{FTB,FTA,FFB,WFN[2^{atom(\Pi)}]\}}(\Pi, \emptyset)$ yield the least fixpoint of Fitting's operator and the well-founded operator, respectively; additional entries over $body(\Pi)$ make valuations of bodies explicit.

3.2.2 Traditional ASP Solvers

We start by investigating the relationship between *smodels* [209] (and *dlv* [158]) on the one hand and our tableau rules on the other hand. After that, we extend these considerations to the rule-based approach of *nomore* [154] as well as to the hybrid assignments dealt with by *nomore++* [3].

The atom-based approach of *smodels* (logically) works on assignments over atoms, and its propagation (cf. [209, 218]) can be specified in terms of the tableau rules shown in Figure 3.3.[5] While *FI* expresses that the head of a rule whose body literals hold must

[5]The names of tableau rules in Figure 3.3 are aligned to the ones used for *smodels*' propagation rules

be true, its contrapositive, *CFH*, describes that a body literal of a rule must not hold if the head is false and all other body literals hold already. Moreover, an unsupported atom p must be false, and *ARC* reflects this by checking for the presence of some body literal that does not hold in every rule with head p. Conversely, *CTH* expresses that the body literals of a rule $p' \leftarrow l'_1, \ldots, l'_i, \ldots, l'_n$ must hold if an atom p is true and any other rule with head p contains some body literal that does not hold.[6] Finally, *AM* formalizes that any atom p belonging to some GRS-unfounded set U, in view of some false body literal in every element of $EB_\Pi(U)$, must be false. Note that Figure 3.3 does not show a contrapositive of *AM*, as *smodels*' propagation does not include such reasoning; it could still be defined analogously to *CTH*, yet requiring the conditions $p \in U$, $U \subseteq atom(\Pi)$, and $\{r \in \Pi \mid head(r) \in U, body(r)^+ \cap U = \emptyset, body(r) \cap \{l_1, \ldots, l_m\} = \emptyset\} \subseteq \{p' \leftarrow l'_1, \ldots, l'_i, \ldots, l'_n\}$ in the proviso (thus checking that every element of $EB_\Pi(U) \setminus \{\{l'_1, \ldots, l'_i, \ldots, l'_n\}\}$ contains some body literal that does not hold), while there is a true atom p in U.

Example 3.4. *Augmenting the tableau rules in Figure 3.3 with Cut*$[atom(\Pi_1)]$*, we can generate the following tableau for* Π_1 *from Example 2.1 and the empty assignment:*

$$\left\{\begin{array}{lll} r_1: & a \leftarrow & \\ r_2: & c \leftarrow not\ b, not\ d & \\ r_3: & d \leftarrow a, not\ c & \end{array}\right\}$$

		$\boldsymbol{T}a$		(*FI*)
		$\boldsymbol{F}b$		(*ARC*)
$\boldsymbol{T}c$			$\boldsymbol{F}c$	($Cut[atom(\Pi_1)]$)
$\boldsymbol{F}d$	(*CTH*)		$\boldsymbol{T}d$	(*CFH*)

This tableau is similar to the one of $\mathcal{T}_{smodels}$ *shown in Figure 3.2, yet it omits entries for rule bodies.*

The next result characterizes *smodels*' propagation in terms of the tableau rules in Figure 3.1.

Proposition 3.3. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment.*
Then, we have that

1. $D_{\{FI\}}(\Pi, \mathbf{A}) = D_{\{FTA\}}(\Pi, D_{\{FTB\}}(\Pi, \mathbf{A}))$;
2. $D_{\{ARC\}}(\Pi, \mathbf{A}) = D_{\{FFA\}}(\Pi, D_{\{FFB\}}(\Pi, \mathbf{A}))$;
3. $D_{\{CTH\}}(\Pi, \mathbf{A}) = D_{\{BTB\}}(\Pi, D_{\{BTA\}}(\Pi, D_{\{FFB\}}(\Pi, \mathbf{A}) \cup \{\boldsymbol{T}p \mid p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)\}))$;
4. $D_{\{CFH\}}(\Pi, \mathbf{A}) = D_{\{BFB\}}(\Pi, D_{\{BFA\}}(\Pi, \mathbf{A}) \cup \{\boldsymbol{T}p \mid p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)\} \cup \{\boldsymbol{F}p \mid p \in \mathbf{A}^{\boldsymbol{F}} \cap atom(\Pi)\})$;
5. $D_{\{AM\}}(\Pi, \mathbf{A}) = D_{\{WFN[2^{atom(\Pi)}]\}}(\Pi, D_{\{FFB\}}(\Pi, \mathbf{A}))$.

in [218], and the tableau rules reflect *smodels*' propagation rules by respective prerequisites and consequents.

[6] If $p' = p$, then $p' \leftarrow l'_1, \ldots, l'_i, \ldots, l'_n$ is the only remaining rule to support p; otherwise, the true atom p is unsupported, and arbitrary body literals may be deduced in the face of a contradiction.

As in Proposition 3.2, *FTB* and *FFB* are used on the right-hand sides to reflect truth or falsity of bodies w.r.t. their contained literals by corresponding entries. Also observe that the characterizations of *FI*, *ARC*, and *AM* are similar to those of $\mathbb{T}_\Pi(\mathbf{A})$, $\mathbb{N}_\Pi(\mathbf{A})$, and $\mathbb{U}_\Pi(\mathbf{A})$ in Proposition 3.2. In addition, backward propagation via *CTH* and *CFH* is captured by means of *BTA* plus *BTB* and *BFA* plus *BFB*, respectively, deducing first entries for bodies that need to be either true or false and then corresponding entries for their contained literals.

In view of the fact that all tableau rules used on the right-hand sides in Proposition 3.3 are contained in tableau calculus $\mathcal{T}_{smodels}$ (along with the observation that their application conditions are monotonic w.r.t. assignments), we immediately conclude the following.

Corollary 3.4. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment.*
Then, we have that $D^*_{\{FI,ARC,CTH,CFH,AM\}}(\Pi, \mathbf{A}) \subseteq D^*_{\mathcal{T}_{smodels}}(\Pi, \mathbf{A})$.

Example 3.5. *To illustrate that the converse of Corollary 3.4 does not hold in general, even if we limit the attention to entries over atoms, consider the following normal program:*

$$\Pi_4 \quad = \quad \left\{ \begin{array}{ll} r_1: & a \leftarrow not\ b \\ r_2: & b \leftarrow not\ a \\ r_3: & c \leftarrow b, not\ a \\ r_4: & d \leftarrow b, not\ a \end{array} \right\}$$

Given $\mathbf{A} = \{\boldsymbol{F}c\}$, *we obtain* $D^*_{\{FI,ARC,CTH,CFH,AM\}}(\Pi_4, \mathbf{A}) = \{\boldsymbol{F}c\}$, *while* $D^*_{\mathcal{T}_{smodels}}(\Pi_4, \mathbf{A}) = \{\boldsymbol{F}c, \boldsymbol{F}\{b, not\ a\}, \boldsymbol{F}d\}$. *Note that the falsity of atom* $c = head(r_3)$ *necessitates the falsity of* $body(r_3) = \{b, not\ a\}$ *because* c *would be derived by* r_3 *otherwise. However, since there is more than one literal in* $\{b, not\ a\}$, *it is not immediately clear which body literal ought to be false. Hence, an "atom-only" approach like the one of smodels gets stuck. In contrast, when we assign the body* $\{b, not\ a\}$ *to false, we see that also the rule* r_4 *with head* d *is inapplicable, which enables* $\mathcal{T}_{smodels}$ *to deduce* $\boldsymbol{F}d$.

We note that, given the similarities between *smodels* and *dlv* on normal programs [122], the latter is also characterized by the tableau rules in Figure 3.3 (along with *Cut*$[atom(\Pi)]$). Interestingly, both *smodels* and *dlv* use dedicated data structures in their implementations for indicating (in)applicability of program rules: in *smodels*, a rule with a false body is marked as "inactive" [208]; similarly, *dlv* determines truth values of bodies from specific counters [64]. It is thus justified to describe the proceeding of atom-based solvers by assignments and tableau rules that incorporate both atoms and bodies.[7] Hence, for comparing atom-based to other ASP solving approaches, we in the following refer to $\mathcal{T}_{smodels}$, rather than a calculus built from the tableau rules in Figure 3.3. As shown in Example 3.5, $\mathcal{T}_{smodels}$ slightly overestimates the propagation capacities of atom-based solvers, so that lower bounds for its efficiency, considered in Section 3.4, also apply to the real solvers, viz., *smodels* and *dlv*.

A similar analysis as in Proposition 3.3 is also possible for the rule-based approach of *nomore*, amounting to assignments over rule bodies, and for the hybrid approach of *nomore++*. By assigning truth values to atoms as well as bodies, the latter works on the same basis as its associated tableau calculus $\mathcal{T}_{nomore++}$. In fact, although we omit

[7] In [124], additional variables for bodies, one for each rule in a program, are explicitly introduced for comparing *smodels* to DPLL, and the characterization of *smodels*' propagation rules in terms of unit propagation provided in [110] likewise shows that rule bodies serve as inherent structural variables.

the details, it is not hard to verify that the propagation operators $\mathcal{P}$, $\mathcal{B}$, and $\mathcal{U}$ (cf. [3]) implemented by *nomore++* match the deterministic tableau rules in $\mathcal{T}_{\textit{nomore++}}$. Furthermore, the choice operator $\mathcal{C}$ of *nomore++* coincides with $\textit{Cut}[atom(\Pi) \cup body(\Pi)]$. On the other hand, when we consider *nomore* augmented with backward propagation [169], we obtain that its propagation is subsumed by $\mathcal{T}_{\textit{nomore}}$. As with $\mathcal{T}_{\textit{smodels}}$ and *smodels*, we have that some entries deducible by $\mathcal{T}_{\textit{nomore}}$ are not inferred by *nomore*.

The following example illustrates that assignments over rule bodies only may be less informative than assignments that also include atoms.

Example 3.6. *Consider the following normal program:*

$$\Pi_5 \;=\; \left\{ \begin{array}{ll} r_1: & a \leftarrow b \\ r_2: & b \leftarrow c \\ r_3: & b \leftarrow not\ c \\ r_4: & c \leftarrow a, not\ b \end{array} \right\}$$

Given $\mathbf{A} = \{\boldsymbol{T}\{b\}\}$*, since two rules,* r_2 *and* r_3*, have* b *as their head, nomore cannot determine a unique program rule to derive* b *from, so that its propagation cannot infer anything. In contrast,* $D^*_{\mathcal{T}_{\textit{nomore}}}(\Pi_5, \mathbf{A}) = \{\boldsymbol{T}\{b\}, \boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}\{a, not\ b\}, \boldsymbol{F}c, \boldsymbol{F}\{c\}, \boldsymbol{T}\{not\ c\}\}$*. Here, the fact that the atom* b *must be true yields that the body* $\{a, not\ b\}$ *is necessarily false, from which* $\mathcal{T}_{\textit{nomore}}$ *then deduces the remaining entries.*

3.2.3 SAT-Based and Conflict-Driven Learning ASP Solvers

Lin and Zhao [167] showed that the answer sets of a normal program Π coincide with the models of the completion of Π satisfying all loop formulas of Π. By introducing auxiliary variables for rule bodies, as also used in [9] to avoid an exponential blow-up due to clausification, the *completion*, $Comp(\Pi)$, and *loop formulas*, $LF(\Pi)$, of a program Π can be defined as follows:

$$\begin{array}{rcl} Comp(\Pi) & = & \{p_B \leftrightarrow (\bigwedge_{p \in B^+} p \wedge \bigwedge_{q \in B^-} \neg q) \mid B \in body(\Pi)\} \\ & \cup & \{p \leftrightarrow (\bigvee_{B \in body_\Pi(p)} p_B) \mid p \in atom(\Pi)\} \\ LF(\Pi) & = & \{(\bigvee_{p \in L} p) \rightarrow (\bigvee_{B \in EB_\Pi(L)} p_B) \mid L \in loop(\Pi)\} \end{array}$$

Given that a program Π may yield exponentially many (non-redundant) loop formulas [164], SAT-based ASP solvers *assat* [167], *cmodels* [123], and *sag* [168] do not a priori construct $LF(\Pi)$. Rather, they use some SAT solver to generate a model of $Comp(\Pi)$ and, afterwards, check whether the model contains a loop L of Π whose loop formula is violated. If so, L is unfounded w.r.t. the model at hand, and adding the loop formula for L eliminates the model as an answer set candidate.[8]

Regarding the generation of answer set candidates, the following analog of Theorem 3.1 applies to $\mathcal{T}_{\textit{comp}}$ and models of $Comp(\Pi)$.

Theorem 3.5. *Let* Π *be a normal program.*

Then, we have that the following holds for tableau calculus $\mathcal{T}_{\textit{comp}}$*:*

1. *Every incomplete tableau for* Π *and* $\emptyset$ *can be extended to a complete tableau for* Π *and* $\emptyset$*.*

[8] While *assat* and *cmodels* apply sophisticated unfounded set checks only w.r.t. total answer set candidates, *sag* can perform them also w.r.t. partial assignments generated by a SAT solver.

2. *$Comp(\Pi)$ has a model X iff every complete tableau for Π and $\emptyset$ has a unique non-contradictory branch $(\Pi, \mathbf{A})$ such that $(\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)) \cup \{p_B \mid B \in \mathbf{A}^{\boldsymbol{T}} \cap body(\Pi)\} = X$.*

3. *$Comp(\Pi)$ has no model iff every complete tableau for Π and $\emptyset$ is a refutation.*

Theorem 3.5 shows that $\mathcal{T}_{comp}$ captures exactly the models of $Comp(\Pi)$ for a normal program Π.[9] Since $\mathcal{T}_{comp}$ admits a non-contradictory complete branch $(\Pi, \mathbf{A})$ in some tableau iff $(\Pi, \mathbf{A})$ belongs to every complete tableau for Π and $\emptyset$, Theorem 3.5 (like Theorem 3.1) remains valid when replacing "every" by "some" in the second and the third item of its statement.

By Theorem 3.1, adding *WFN*$[2^{atom(\Pi)}]$ to $\mathcal{T}_{comp}$ leads to a tableau calculus characterizing answer sets. However, SAT-based ASP solvers check for unfounded loops rather than arbitrary unfounded sets, while atom-wise unfounded set handling is accomplished via *FFA* (enclosed in $Comp(\Pi)$). Given that *FFA* is subsumed by *WFN*$[2^{atom(\Pi)}]$, but not by *WFN*$[loop(\Pi)]$, the next result, which shows that *WFN*$[2^{atom(\Pi)}]$ and *WFN*$[loop(\Pi)]$ plus *FFA* yield the same deductive closure (in combination with *FFB*, deducing the falsity of bodies from body literals that do not hold), tells us that limiting *WFN* to loops abolishes overlaps between tableau rules.

Proposition 3.6. *Let Π be a normal program and $\mathbf{A}$ an assignment.*
Then, we have that $D^*_{\{FFB, WFN[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) = D^*_{\{FFB, FFA, WFN[loop(\Pi)]\}}(\Pi, \mathbf{A})$.

Proposition 3.6 is obtained from Theorem 2.9, which showed that any unfounded set containing some non-false atom also includes an unfounded loop of non-false atoms, provided that the underlying assignment is both atom- and body-saturated. The following auxiliary result states the fact that both properties hold w.r.t. any assignment closed under *FFB* and *FFA*.

Lemma 3.7. *Let Π be a normal program and $\mathbf{A}$ an assignment.*
Then, we have that

1. *$\mathbf{A}$ is body-saturated for Π iff $D_{\{FFB\}}(\Pi, \mathbf{A}) \subseteq \mathbf{A}$;*

2. *$\mathbf{A}$ is atom-saturated for Π iff $D_{\{FFA\}}(\Pi, \mathbf{A}) \subseteq \mathbf{A}$.*

In view of Proposition 3.6, by adding the tableau rule *WFN*$[loop(\Pi)]$ to $\mathcal{T}_{comp}$, we characterize models of $Comp(\Pi) \cup LF(\Pi)$, which coincide with the answer sets of Π.

Theorem 3.8. *Let Π be a normal program.*
Then, we have that the following holds for tableau calculus $\mathcal{T}_{comp} \cup \{WFN[loop(\Pi)]\}$:

1. *Every incomplete tableau for Π and $\emptyset$ can be extended to a complete tableau for Π and $\emptyset$.*

2. *Program Π has an answer set X iff every complete tableau for Π and $\emptyset$ has a unique non-contradictory branch $(\Pi, \mathbf{A})$ such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$.*

3. *Program Π has no answer set iff every complete tableau for Π and $\emptyset$ is a refutation.*

As with Theorem 3.1 and 3.5, we have that Theorem 3.8 remains valid when replacing "every" by "some" in the second and the third item of its statement.

[9] The models of the completion of a logic program Π are also called the "supported models" of Π [6].

Example 3.7. *Consider the following normal program:*

$$\Pi_6 \;=\; \left\{ \begin{array}{ll} r_1: & a \leftarrow not\ b \\ r_2: & b \leftarrow not\ a \\ r_3: & c \leftarrow a \\ r_4: & c \leftarrow d \\ r_5: & d \leftarrow c, not\ a \\ r_6: & e \leftarrow c \\ r_7: & e \leftarrow d \end{array} \right\}$$

We have that $loop(\Pi_6) = \{\{c,d\}\}$ *and* $EB_{\Pi_6}(\{c,d\}) = EB_{\Pi_6}(\{c,d,e\}) = \{\{a\}\}$. *For* $\mathbf{A} = \{\boldsymbol{F}\{a\}\}$, *we thus get* $D_{\{WFN[2^{atom(\Pi_6)}]\}}(\Pi_6, \mathbf{A}) = \{\boldsymbol{F}c, \boldsymbol{F}d, \boldsymbol{F}e\}$, *while* $D_{\{WFN[loop(\Pi_6)]\}}(\Pi_6, \mathbf{A}) = \{\boldsymbol{F}c, \boldsymbol{F}d\}$ *does not include* $\boldsymbol{F}e$. *As* $body_{\Pi_6}(e) = \{\{c\},\{d\}\} \subseteq \big(D_{\{FFB\}}(\Pi_6, \{\boldsymbol{F}c, \boldsymbol{F}d\})\big)^{\boldsymbol{F}}$, $\boldsymbol{F}e \in D^*_{\{FFB,FFA,WFN[loop(\Pi_6)]\}}(\Pi_6, \mathbf{A})$ *nonetheless holds. Hence, the combination of FFB, FFA, and WFN*$[loop(\Pi_6)]$ *allows us to deduce the same entries as obtained with FFB and WFN*$[2^{atom(\Pi_6)}]$.

To see the relationship between the tableau rules in Figure 3.1 and (Boolean) constraints induced by $Comp(\Pi) \cup LF(\Pi)$, which are further investigated in Chapter 4, note that a (deterministic) tableau rule with prerequisites $\sigma_1, \ldots, \sigma_n$ and consequent σ intrinsically expresses the fact that $\{\sigma_1, \ldots, \sigma_n, \overline{\sigma}\}$ is a nogood.

Example 3.8. *For* Π_6 *from Example 3.7 and the tableau rules in* $\mathcal{T}_{comp}$, *the following set* Δ *constitutes the union of (unsubsumed) nogoods given by tableau rules (other than Cut*$[atom(\Pi_6) \cup body(\Pi_6)]$*):*

$$\Delta \;=\; \left\{ \begin{array}{l} \{\boldsymbol{F}\{not\ b\}, \boldsymbol{F}b\}, \{\boldsymbol{F}\{not\ a\}, \boldsymbol{F}a\}, \{\boldsymbol{F}\{a\}, \boldsymbol{T}a\}, \{\boldsymbol{F}\{d\}, \boldsymbol{T}d\}, \\ \{\boldsymbol{F}\{c\}, \boldsymbol{T}c\}, \{\boldsymbol{F}\{c, not\ a\}, \boldsymbol{T}c, \boldsymbol{F}a\} \end{array} \right\} \quad (3.1)$$

$$\cup \left\{ \begin{array}{l} \{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}, \{\boldsymbol{T}\{not\ a\}, \boldsymbol{T}a\}, \{\boldsymbol{T}\{a\}, \boldsymbol{F}a\}, \{\boldsymbol{T}\{d\}, \boldsymbol{F}d\}, \\ \{\boldsymbol{T}\{c\}, \boldsymbol{F}c\}, \{\boldsymbol{T}\{c, not\ a\}, \boldsymbol{F}c\}, \{\boldsymbol{T}\{c, not\ a\}, \boldsymbol{T}a\} \end{array} \right\} \quad (3.2)$$

$$\cup \left\{ \begin{array}{l} \{\boldsymbol{F}a, \boldsymbol{T}\{not\ b\}\}, \{\boldsymbol{F}b, \boldsymbol{T}\{not\ a\}\}, \{\boldsymbol{F}c, \boldsymbol{T}\{a\}\}, \{\boldsymbol{F}c, \boldsymbol{T}\{d\}\}, \\ \{\boldsymbol{F}d, \boldsymbol{T}\{c, not\ a\}\}, \{\boldsymbol{F}e, \boldsymbol{T}\{c\}\}, \{\boldsymbol{F}e, \boldsymbol{T}\{d\}\} \end{array} \right\} \quad (3.3)$$

$$\cup \left\{ \begin{array}{l} \{\boldsymbol{T}a, \boldsymbol{F}\{not\ b\}\}, \{\boldsymbol{T}b, \boldsymbol{F}\{not\ a\}\}, \{\boldsymbol{T}c, \boldsymbol{F}\{a\}, \boldsymbol{F}\{d\}\}, \\ \{\boldsymbol{T}d, \boldsymbol{F}\{c, not\ a\}\}, \{\boldsymbol{T}e, \boldsymbol{F}\{c\}, \boldsymbol{F}\{d\}\} \end{array} \right\} \quad (3.4)$$

The nogoods in (3.1) are obtained from tableau rule FTB, and likewise from its contrapositive BFB. Similarly, the nogoods in (3.2), (3.3), and (3.4) result from FFB or BTB, FTA or BFA, and FFA or BTA, respectively. Note that a solution $\mathbf{A}$ *for the nogoods in* Δ *coincides with the model* $(\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi_6)) \cup \{p_B \mid B \in \mathbf{A}^{\boldsymbol{T}} \cap body(\Pi_6)\}$ *of* $Comp(\Pi_6)$. *Since the nogoods in (3.1)–(3.4) can easily be represented by clauses, e.g.,* $a \vee \neg p_{\{not\ b\}}$ *for* $\{\boldsymbol{F}a, \boldsymbol{T}\{not\ b\}\}$, *the (deterministic) tableau rules in* $\mathcal{T}_{comp}$ *can be used to derive the input for a SAT solver, as engaged by assat, cmodels, and sag. In turn, clauses stemming from completion (cf. [9]) represent nogoods similar to the ones in* Δ.

Example 3.9. *When we extract nogoods from tableau rule WFN*$[loop(\Pi_6)]$, *or likewise from WFJ*$[loop(\Pi_6)]$, *we get the set* $\Lambda = \{\{\boldsymbol{T}c, \boldsymbol{F}\{a\}\}, \{\boldsymbol{T}d, \boldsymbol{F}\{a\}\}\}$ *of nogoods. Solutions for* Λ *coincide with the models of* $LF(\Pi_6) = \{(c \vee d) \rightarrow p_{\{a\}}\}$. *Once clauses representing* Λ, *such as* $\neg c \vee p_{\{a\}}$ *and* $\neg d \vee p_{\{a\}}$, *have been added to those representing* Δ *from Example 3.8 (in order to eliminate some invalid answer set candidate), they can be used by a SAT solver.*

In native conflict-driven learning ASP solvers, such as *smodels*$_{cc}$ [218] and *clasp* [95], the nogoods obtained from tableau rules contribute reasons for conflicts. For *clasp*, such nogoods correspond to the sets Δ_Π and Λ_Π (cf. [95]), which are developed in Section 4.1, and the (implicit) constraints propagated by *smodels*$_{cc}$, extracting reasons relative to *smodels*' propagation rules, are closely related to nogoods obtained from tableau rules (cf. [124, 110, 122]). This indicates that the tableau rules in Figure 3.1 are well-suited to characterize conditions for propagation as applied in both SAT-based and native conflict-driven learning ASP solvers. Of course, our tableaux reflect neither preprocessing techniques, such as the ones described in [9, 96], nor backjumping and learning schemes of conflict-driven learning SAT and ASP solvers (see, e.g., [179, 219, 56, 21]). We note that the calculus based on state transition graphs presented in [160] captures backjumping and conflict-driven learning as performed by *smodels*$_{cc}$; in that work, the clauses attributed to *smodels*$_{cc}$ avoid auxiliary variables for rule bodies in favor of (non-erasable) duplicate literals. Unlike this, in Chapter 4, we devise algorithms that combine the propagation conditions expressed by tableau rules with backjumping and conflict-driven learning.

Finally, let us comment on some particularities of unfounded set handling. On the one hand, Proposition 3.6 shows that tableau rule *WFN*$[2^{atom(\Pi)}]$ can be replaced by more restrictive rule *WFN*$[loop(\Pi)]$ without sacrificing deducible entries. In fact, SAT-based ASP solvers concentrate the consideration of positive recursion on loops, and native ASP solvers like *clasp*, *dlv*, and *smodels* exploit strongly connected components of programs' dependency graphs to achieve a similar effect. However, no existing ASP solver incorporates a contrapositive of *WFN*, that is, *WFJ*$[2^{atom(\Pi)}]$ or *WFJ*$[loop(\Pi)]$, in its propagation (unless loop formulas have been recorded). An approach to extend unfounded set handling in this direction has been presented in [33, 34]. Unfortunately, it amounts to failed-literal detection (cf. [76, 209]), whose high (polynomial) computational cost makes its unrestricted application prohibitive in practice. It is still interesting that a result similar to Proposition 3.6 cannot be obtained for *WFJ*$[2^{atom(\Pi)}]$ and *WFJ*$[loop(\Pi)]$.

Example 3.10. *For* Π_6 *from Example 3.7, we have that WFJ*$[loop(\Pi_6)]$ *is strictly weaker than WFJ*$[2^{atom(\Pi_6)}]$. *Given* $\mathbf{A} = \{\boldsymbol{T}e\}$, *since* $EB_{\Pi_6}(\{c,d,e\}) = \{\{a\}\}$, *we obtain that* $\boldsymbol{T}\{a\} \in D_{\{WFJ[2^{atom(\Pi_6)}]\}}(\Pi_6, \mathbf{A})$. *On the other hand, since* $loop(\Pi_6) = \{\{c,d\}\}$ *and* $body_{\Pi_6}(e) = \{\{c\},\{d\}\}$, *neither WFJ*$[loop(\Pi_6)]$ *nor any of the tableau rules (a)–(i) in Figure 3.1 (in particular, BTA) is applicable in the branch* $(\Pi_6, \mathbf{A})$, *that is,* $D_{\{(a)\text{–}(i),WFJ[loop(\Pi_6)]\}}(\Pi_6, \mathbf{A}) = \emptyset$.

Regarding the impact of not at all applying *WFJ* or restricting the sets of atoms to which it can be applied to loops, in Section 3.4, we show that *WFJ* can be simulated by means of *Cut* and *WFN*, using exactly the idea of failed-literal detection.

3.3 Generic Tableaux for Composite Language Constructs

In what follows, we generalize our approach and develop an extensible tableau framework for logic programs incorporating composite language constructs, such as *dlv*'s aggregates [67] or *smodels*' cardinality and weight constraints [209]. To this end, we take a more abstract perspective than before and view, e.g., conjunctions as (simple) Boolean aggregates, which like atoms can be preceded by not. However, we also show below that the dedicated tableau framework introduced in Section 3.1 and the generic approach developed in the sequel coincide on the common language fragment of normal programs.

The basic idea of our generic approach is to specify core tableau rules that, for one, aim at establishing model conditions by propagating truth and falsity of entries along program rules.

Example 3.11. *Consider the rule* $0\{a,d\}1 \leftarrow 1\{b, not\ e\}2$*, including two cardinality constraints.*[10] *If* $1\{b, not\ e\}2$ *holds w.r.t. an assignment, we know that* $0\{a,d\}1$ *must hold as well; conversely,* $1\{b, not\ e\}2$ *must not hold if it is known that* $0\{a,d\}1$ *does not hold. In our generic framework, such inferences are reflected by deducing* $\boldsymbol{T}0\{a,d\}1$ *from* $\boldsymbol{T}1\{b, not\ e\}2$ *or, conversely,* $\boldsymbol{F}1\{b, not\ e\}2$ *from* $\boldsymbol{F}0\{a,d\}1$.

Notice that we associate truth values with cardinality constraints, and that matching them to truth values of the cardinality constraints' literals requires specific tableau rules. Hence, when we below augment our framework with composite language constructs, we also introduce such construct-specific tableau rules.

Regardless of concrete program syntax, an answer set must be a minimal model of the reduct relative to itself, as in the case of normal programs (cf. Section 2.1). Our generic tableau rules reflect this minimality requirement by investigating external supports of sets of atoms. To this end, for a rule $\alpha \leftarrow \beta$ and an assignment $\mathbf{A}$, we make use of two predicates, $\overleftarrow{sup}_{\mathbf{A}}(\alpha, S)$ and $\overrightarrow{sup}_{\mathbf{A}}(\beta, S')$, to check whether a set S of atoms can be supported via α and whether β can hold independently of atoms in S'. Since our tableau rules consider subsets S' of S, viz., $S' = \emptyset$ or $S' = S$, the two predicates allow us to test whether $\alpha \leftarrow \beta$ provides an external support for S w.r.t. $\mathbf{A}$. Of particular interest are the cases where there is no external support for S, as it tells us that all atoms of S must be false, or where there is exactly one external support $\alpha \leftarrow \beta$ for S. In the latter case, if S contains some true atom, we know that β must hold, and additional entries might be required too, for which we provide two sets, $min_{\mathbf{A}}(\alpha, S)$ and $max_{\mathbf{A}}(\beta, S')$.

Example 3.12. *If at least one of the entries* $\boldsymbol{T}a$ *and* $\boldsymbol{T}b$ *belongs to a given assignment* $\mathbf{A}$ *and the rule* $0\{a,d\}1 \leftarrow 1\{b, not\ e\}2$ *is the single external support for the set* $\{a,b\}$*, we get* $min_{\mathbf{A}}(0\{a,d\}1, \{a,b\}) = \{\boldsymbol{F}d\}$ *and* $max_{\mathbf{A}}(1\{b, not\ e\}2, \{a,b\}) = \{\boldsymbol{F}e\}$*. In fact, if* d *was true, the upper bound* 1 *of* $0\{a,d\}1$ *would be reached, so that no further atom can be supported. Likewise, if the literal* $not\ e$ *in* $1\{b, not\ e\}2$ *was false, the lower bound* 1 *of* $1\{b, not\ e\}2$ *could only be achieved by making* b *true, so that the support is not external to* $\{a,b\}$.

Since the definitions of $\overleftarrow{sup}$, $\overrightarrow{sup}$, min, and max are specific to a language construct at hand, we gradually extend them below when integrating composite language constructs into our generic framework. To be more precise, after in Section 3.3.1 generalizing the definitions from Section 2.1, we devise generic tableau rules in Section 3.3.2. These are augmented with tableau rules for conjunctions, cardinality constraints, and disjunctions in Section 3.3.3, 3.3.4, and 3.3.5, respectively.

3.3.1 Answer Sets for Propositional Theories

Among several proposals defining answer sets for logic programs accommodating particular language extensions (e.g., [209, 72, 65, 69, 171, 66]), we rely on the one by

[10] A cardinality constraint like $1\{b, not\ e\}2$ resembles a linear inequality, viz., $1 \leq b + (1 - e) \leq 2$, over Boolean variables. By assigning each of the variables b and e to either 1 (*true*) or 0 (*false*), the inequality evaluates to *true* or *false*, respectively. For instance, $1 \leq 1 + (1 - 1) \leq 2$ holds with $b = e = 1$, while $1 \leq 0 + (1 - 1) \leq 2$, obtained with $b = 0$ and $e = 1$, does not hold.

Ferraris [69], as it is general enough to deal with arbitrary (ground) aggregates and has a firm basis in the logic of here-and-there [192, 163]. To achieve generality, this semantics applies to propositional theories and identifies aggregates with propositional formulas.

Formally, a propositional theory is a finite set of propositional formulas, constructed from atoms in a denumerable alphabet $\mathcal{P}$ and the connectives $\bot$, $\wedge$, $\vee$, and $\rightarrow$. Any other connective is considered as an abbreviation, in particular, $\neg\phi$ stands for $\phi \rightarrow \bot$. An interpretation, represented by the set X of its entailed atoms, is a model of a propositional theory Φ if $X \models \phi$ for all $\phi \in \Phi$, where $\models$ is the standard satisfaction relation of propositional logic. The *reduct*, denoted by Φ^X, of Φ w.r.t. X is a propositional theory, (recursively) defined as follows:

$$\begin{array}{rcl} \Phi^X & = & \{\phi^X \mid \phi \in \Phi\} \\ \phi^X & = & \begin{cases} \bot & \text{if } X \not\models \phi \\ \phi & \text{if } \phi \in X \\ \phi_1^X \circ \phi_2^X & \text{if } X \models \phi \text{ and } \phi = \phi_1 \circ \phi_2 \text{ for } \circ \in \{\wedge, \vee, \rightarrow\} \end{cases} \end{array}$$

Intuitively, all (maximal) subformulas of Φ that are false in X are replaced by $\bot$ in Φ^X, while other subformulas of Φ stay intact. Hence, any model X of Φ is a model of Φ^X as well. Also note that all occurrences of negation, i.e., subformulas of the form $\phi \rightarrow \bot$, are replaced by constants in Φ^X, since either ϕ or $\phi \rightarrow \bot$ is false in X. An interpretation X is an *answer set* of a propositional theory Φ if X is a minimal model of Φ^X. In fact, an answer set X of Φ is the unique least model of Φ^X because all atoms occurring in Φ^X belong to X, but a least model of Φ^X is, in general, not guaranteed to exist.

Example 3.13. *Consider the following propositional theory:*

$$\Phi \;=\; \{\; \phi\colon \;\; a \vee b \;\}$$

We obtain the following reducts of its single formula ϕ w.r.t. the four subsets of $\{a, b\}$:

$$\begin{array}{rcl} \phi^{\emptyset} & = & \bot \\ \phi^{\{a\}} & = & a \vee \bot \\ \phi^{\{b\}} & = & \bot \vee b \\ \phi^{\{a,b\}} & = & a \vee b \end{array}$$

While $\emptyset$ is not a model of $\phi^{\emptyset}$, the proper subsets $\{a\}$ and $\{b\}$ of $\{a, b\}$ are models of $\phi^{\{a,b\}}$. Hence, neither $\emptyset$ nor $\{a, b\}$ is an answer set of Φ. On the other hand, $\{a\}$ and $\{b\}$ are minimal models of $\phi^{\{a\}}$ and $\phi^{\{b\}}$, respectively. That is, $\{a\}$ and $\{b\}$ are answer sets of Φ. Furthermore, observe that a is the only atom occurring in $\phi^{\{a\}}$, and the same applies to b and $\phi^{\{b\}}$.

In a general setting, we understand a (*propositional*) *logic program* Π over a denumerable alphabet $\mathcal{P}$ as a finite set of rules of the form $\alpha \leftarrow \beta$, where α and β are *literals*, that is, expressions over $\mathcal{P}$ possibly preceded by not. As before, $atom(\Pi)$ denotes the set of atoms occurring in Π. In the following, we refine heads α and bodies β for obtaining particular classes of logic programs. The semantics of a logic program is given by the answer sets of an associated propositional theory, obtained via a translation τ described below. However, the proof-theoretic characterizations we provide apply directly to logic programs, without translating them to propositional theories.

3.3.2 Generic Tableau Rules

We begin with a simple class of *unary programs* where rules $\alpha \leftarrow \beta$ are restricted to *atomic literals*, that is, each of α and β is equal to either p or $not\ p$ for an atom $p \in \mathcal{P}$.[11] The semantics of a unary program Π is given by the answer sets of a propositional theory, $\tau[\Pi]$, (recursively) defined as follows:

$$\tau[\Pi] = \{\tau[\beta] \rightarrow \tau[\alpha] \mid (\alpha \leftarrow \beta) \in \Pi\} \tag{3.5}$$

$$\tau[\pi] = \begin{cases} \neg\tau[v] & \text{if } \pi = not\ v \\ \pi & \text{if } \pi \in \mathcal{P} \end{cases} \tag{3.6}$$

Example 3.14. *Consider the following unary program:*

$$\Pi_7 = \left\{ \begin{array}{ll} r_1: & a \leftarrow not\ b \\ r_2: & not\ a \leftarrow c \\ r_3: & b \leftarrow c \\ r_4: & c \leftarrow b \end{array} \right\}$$

The associated propositional theory is as follows:

$$\tau[\Pi_7] = \left\{ \begin{array}{ll} \tau[r_1]: & \neg b \rightarrow a \\ \tau[r_2]: & c \rightarrow \neg a \\ \tau[r_3]: & c \rightarrow b \\ \tau[r_4]: & b \rightarrow c \end{array} \right\}$$

The sets $\{a\}$ and $\{b, c\}$ are models of $\tau[\Pi_7]$, and their respective reducts are:

$$(\tau[\Pi_7])^{\{a\}} = \left\{ \begin{array}{ll} (\tau[r_1])^{\{a\}}: & \neg\bot \rightarrow a \\ (\tau[r_2])^{\{a\}}: & \bot \rightarrow \bot \\ (\tau[r_3])^{\{a\}}: & \bot \rightarrow \bot \\ (\tau[r_4])^{\{a\}}: & \bot \rightarrow \bot \end{array} \right\}$$

$$(\tau[\Pi_7])^{\{b,c\}} = \left\{ \begin{array}{ll} (\tau[r_1])^{\{b,c\}}: & \bot \rightarrow \bot \\ (\tau[r_2])^{\{b,c\}}: & c \rightarrow \neg\bot \\ (\tau[r_3])^{\{b,c\}}: & c \rightarrow b \\ (\tau[r_4])^{\{b,c\}}: & b \rightarrow c \end{array} \right\}$$

Clearly, $\{a\}$ is the least model of $(\tau[\Pi_7])^{\{a\}}$, so that $\{a\}$ is an answer set of Π_7. The (unique) minimal model of $(\tau[\Pi_7])^{\{b,c\}}$ is $\emptyset$. Hence, $\{b, c\}$ is not the least model of $(\tau[\Pi_7])^{\{b,c\}}$ and thus not an answer set of Π_7.

While the semantics is based on translation to propositional theories, our tableau framework deals with logic programs as such. The global design, however, follows the two semantic requirements for answer sets: modelhood w.r.t. a program and (non-circular) support w.r.t. the reduct. In order to establish the latter, for a program Π, two sets $S \subseteq atom(\Pi), S' \subseteq atom(\Pi)$, and an assignment $\mathbf{A}$, we define:

$$sup_{\mathbf{A}}(\Pi, S, S') = \left\{(\alpha \leftarrow \beta) \in \Pi \mid \boldsymbol{f}\beta \notin \mathbf{A},\ \overleftarrow{sup}_{\mathbf{A}}(\alpha, S),\ \overrightarrow{sup}_{\mathbf{A}}(\beta, S')\right\} \tag{3.7}$$

The purpose of $sup_{\mathbf{A}}(\Pi, S, S')$ is to determine all rules of Π that can, w.r.t. $\mathbf{A}$, provide a support for the atoms in S that is external to S'. Of particular interest are the cases

[11] Our notion of a unary program is different from the one considered in [136]. The latter allows for one positive and arbitrarily many negative body literals, but not for negative head literals.

where $sup_{\mathbf{A}}(\Pi, S, S')$ is empty or a singleton $\{\alpha \leftarrow \beta\}$. In the first case, the atoms in S cannot be supported and are prone to be false, while the second case tells us that $\alpha \leftarrow \beta$ is the unique support for S external to S'. Since we below consider only situations where $S' \subseteq S$, viz., $S' = \emptyset$ and $S' = S$, $sup_{\mathbf{A}}(\Pi, S, S') = \{\alpha \leftarrow \beta\}$ indicates that β must hold to (non-circularly) support S.

Further investigating the definition of $sup_{\mathbf{A}}(\Pi, S, S')$ in (3.7), we note that a rule $\alpha \leftarrow \beta$ such that $\boldsymbol{f}\beta \in \mathbf{A}$ cannot provide any support w.r.t. $\mathbf{A}$. Otherwise, we check via $\overleftarrow{sup}_{\mathbf{A}}(\alpha, S)$ that α can support S, and via $\overrightarrow{sup}_{\mathbf{A}}(\beta, S')$ that β does not (positively) rely on S'. For the simple class of unary programs, these concepts are defined as follows:

$$\overleftarrow{sup}_{\mathbf{A}}(p, S) \qquad \text{if } p \in S \tag{3.8}$$

$$\overrightarrow{sup}_{\mathbf{A}}(p, S') \qquad \text{if } p \in \mathcal{P} \setminus S' \tag{3.9}$$

$$\overrightarrow{sup}_{\mathbf{A}}(not\ v, S') \qquad \text{for every expression } v \tag{3.10}$$

The universal validity of (3.10) is because only *positive* dependencies are taken into account. Also note that a rule $\alpha \leftarrow \beta$ such that $\alpha = not\ v$ cannot support any set S of atoms. (We further illustrate the above concepts below Theorem 3.9.)

The tableau rules constituting our primal generic calculus are shown in Figure 3.4. Among them, $I{\uparrow}$ and $I{\downarrow}$ provide *rule-based* inferences, such as modus ponens and modus tollens. The tableau rules $N{\uparrow}$ and $N{\downarrow}$ amount to *negation* and *support* for atoms, building on similar principles as completion (of normal programs). Note that the derivability of an atom p and thus the applicability of tableau rules $N{\uparrow}$ and $N{\downarrow}$, respectively, is determined by $sup_{\mathbf{A}}(\Pi, \{p\}, \emptyset)$. In the general case, rule $N{\downarrow}$ makes use of two further concepts, $min_{\mathbf{A}}(\alpha, S)$ and $max_{\mathbf{A}}(\beta, S')$, used to determine entries that must necessarily be added to $\mathbf{A}$ in order to support some atom in S via $\alpha \leftarrow \beta$ without positively relying on S'. However, these concepts play no role in the setting of unary programs:

$$min_{\mathbf{A}}(p, S) = \emptyset \quad \text{for } p \in \mathcal{P} \tag{3.11}$$

$$max_{\mathbf{A}}(p, S') = \emptyset \quad \text{for } p \in \mathcal{P} \tag{3.12}$$

$$max_{\mathbf{A}}(not\ v, S') = \emptyset \quad \text{for every expression } v \tag{3.13}$$

Furthermore, the tableau rules $U{\uparrow}$ and $U{\downarrow}$ take care of (non-empty) "unfounded sets," either by identifying atoms that cannot be non-circularly supported ($U{\uparrow}$) or by preventing true atoms from becoming unfounded ($U{\downarrow}$). The applicability of $U{\uparrow}$ and $U{\downarrow}$ is determined by $sup_{\mathbf{A}}(\Pi, S, S)$ for a set S of atoms. Since $sup_{\mathbf{A}}(\Pi, S, S) \subseteq sup_{\mathbf{A}}(\Pi, S, S')$ for every $S' \subseteq S$ (cf. (3.9)) and, in particular, for $S' = \emptyset$, $U{\uparrow}$ and $U{\downarrow}$ subsume $N{\uparrow}$ and $N{\downarrow}$, which rely on the weaker concept $sup_{\mathbf{A}}(\Pi, \{p\}, \emptyset)$. We nonetheless include $N{\uparrow}$ and $N{\downarrow}$ because their applicability is easy to determine, and thus they have counterparts in virtually all ASP solvers. Also note that we do not parameterize $U{\uparrow}$ and $U{\downarrow}$ by Ω, which has been included in corresponding tableau rules *WFN*$[\Omega]$ and *WFJ*$[\Omega]$ in Figure 3.1 on Page 19 to reflect a possible restriction to loops. Albeit loops can also be identified in propositional theories [70], the generalization is not straightforward and omitted here for brevity. Finally, the *Cut*$[\Gamma]$ rule, allowing for case analyses on the expressions in Γ, is identical to its counterpart in Figure 3.1.

For a unary program Π, we fix the domain of assignments $\mathbf{A}$ as well as the cut objects used by *Cut*$[\Gamma]$ to $dom(\mathbf{A}) = \Gamma = atom(\Pi)$. Similar to Theorem 3.1 on Page 20 applying to normal programs, we can now characterize the answer sets of unary programs in terms of generic tableaux.

$$\frac{\alpha \leftarrow \beta \quad \boldsymbol{t}\beta}{\boldsymbol{t}\alpha}$$

(a) Implication ($I\uparrow$)

$$\frac{\alpha \leftarrow \beta \quad \boldsymbol{f}\alpha}{\boldsymbol{f}\beta}$$

(b) Contraposition ($I\downarrow$)

$$\frac{\Pi, \mathbf{A}}{\boldsymbol{F}p} \ \big(p \in atom(\Pi), sup_{\mathbf{A}}(\Pi, \{p\}, \emptyset) = \emptyset\big)$$

(c) Negation ($N\uparrow$)

$$\frac{\Pi, \mathbf{A}}{\boldsymbol{t}\beta, min_{\mathbf{A}}(\alpha, \{p\}), max_{\mathbf{A}}(\beta, \emptyset)} \ \big(p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi), sup_{\mathbf{A}}(\Pi, \{p\}, \emptyset) = \{\alpha \leftarrow \beta\}\big)$$

(d) Support ($N\downarrow$)

$$\frac{\Pi, \mathbf{A}}{\boldsymbol{F}p} \ \big(S \subseteq atom(\Pi), p \in S, sup_{\mathbf{A}}(\Pi, S, S) = \emptyset\big)$$

(e) Unfounded Set ($U\uparrow$)

$$\frac{\Pi, \mathbf{A}}{\boldsymbol{t}\beta, min_{\mathbf{A}}(\alpha, S), max_{\mathbf{A}}(\beta, S)} \ \big(S \subseteq atom(\Pi), \mathbf{A}^{\boldsymbol{T}} \cap S \neq \emptyset, sup_{\mathbf{A}}(\Pi, S, S) = \{\alpha \leftarrow \beta\}\big)$$

(f) Well-Founded Set ($U\downarrow$)

$$\frac{}{\boldsymbol{T}v \quad | \quad \boldsymbol{F}v} \ (v \in \Gamma)$$

(g) Cut (Cut$[\Gamma]$)

Figure 3.4: Tableau rules for rules *(a),(b)*; atoms *(c),(d)*; sets of atoms *(e),(f)*; and cutting *(g)*.

Theorem 3.9. *Let* Π *be a unary program.*

Then, we have that the following holds for the tableau calculus consisting of the tableau rules (a)–(g):

1. *Every incomplete tableau for* Π *and* $\emptyset$ *can be extended to a complete tableau for* Π *and* $\emptyset$*.*

2. *Program* Π *has an answer set* X *iff every complete tableau for* Π *and* $\emptyset$ *has a unique non-contradictory branch* $(\Pi, \mathbf{A})$ *such that* $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$*.*

3. *Program* Π *has no answer set iff every complete tableau for* Π *and* $\emptyset$ *is a refutation.*

As with the tableau rules in Figure 3.1, we have that the generic calculus including the tableau rules *(a)–(g)* admits a (unique) non-contradictory complete branch $(\Pi, \mathbf{A})$ in some tableau iff $(\Pi, \mathbf{A})$ belongs to every complete tableau for Π and $\emptyset$. Hence, Theorem 3.9 as well as its generalizations to further language constructs provided in the sequel remain

valid when replacing "every" by "some" in their second and their third item.[12] Yet before turning to extensions, let us illustrate the generic tableau rules in Figure 3.4 on a couple of examples.

Example 3.15. *Consider a tableau with* $\{a \leftarrow not\ a\}$ *at its root. A cut on* a *yields two branches, one with* $\boldsymbol{T}a$ *and another one with* $\boldsymbol{F}a$*. The first branch can be closed by deducing* $\boldsymbol{F}a$ *via* $N\uparrow$*. To see this, observe that* $sup_{\{\boldsymbol{T}a\}}(\{a \leftarrow not\ a\}, \{a\}, \emptyset) = \emptyset$ *because* $\boldsymbol{f}not\ a = \boldsymbol{T}a \in \{\boldsymbol{T}a\}$*. That is,* a *must be false since all rules from which it could be derived are inapplicable. The second branch can be closed by deducing* $\boldsymbol{T}a$ *via* $I\uparrow$*, given that* $\boldsymbol{t}not\ a = \boldsymbol{F}a \in \{\boldsymbol{F}a\}$*. Hence, all branches of the tableau are contradictory, indicating that the unary program* $\{a \leftarrow not\ a\}$ *has no answer set.*

Example 3.16. *For another example, consider the following unary program:*

$$\Pi_8 \quad = \quad \left\{ \begin{array}{ll} r_1: & a \leftarrow not\ b \\ r_2: & b \leftarrow not\ a \\ r_3: & c \leftarrow not\ a \end{array} \right\}$$

Cutting on c *results in branches with* $\boldsymbol{T}c$ *and* $\boldsymbol{F}c$*, respectively. The first one can be extended by* $\boldsymbol{t}not\ a = \boldsymbol{F}a$ *via* $N\downarrow$*. Indeed,* $sup_{\{\boldsymbol{T}c\}}(\Pi_8, \{c\}, \emptyset) = \{c \leftarrow not\ a\}$ *tells us that* r_3 *is the only rule that allows for deriving* c*, which necessitates* a *to be false. To be more precise, we have that* $\boldsymbol{f}not\ a = \boldsymbol{T}a \notin \{\boldsymbol{T}c\}$*, and both* $\overleftarrow{sup}_{\{\boldsymbol{T}c\}}(c, \{c\})$ *and* $\overrightarrow{sup}_{\{\boldsymbol{T}c\}}(not\ a, \emptyset)$ *are satisfied. This shows that the proviso of* $N\downarrow$ *is established, so that we can deduce* $\boldsymbol{t}not\ a = \boldsymbol{F}a$*. Given* $\boldsymbol{F}a$*, we can further apply* $I\uparrow$ *or* $I\downarrow$ *to deduce* $\boldsymbol{T}b$ *and to so obtain a non-contradictory complete branch. The second branch with* $\boldsymbol{F}c$ *can be extended by* $\boldsymbol{f}not\ a = \boldsymbol{T}a$ *via* $I\downarrow$*. Given* $\boldsymbol{T}a$*, we deduce* $\boldsymbol{F}b$*, by* $N\uparrow$ *or* $N\downarrow$*, to obtain a second non-contradictory complete branch. The two complete branches, consisting of* Π_8 *along with* $\{\boldsymbol{T}c, \boldsymbol{F}a, \boldsymbol{T}b\}$ *and* $\{\boldsymbol{F}c, \boldsymbol{T}a, \boldsymbol{F}b\}$*, respectively, tell us that* $\{b, c\}$ *and* $\{a\}$ *are the two answer sets of* Π_8*.*

Example 3.17. *Finally, consider the following unary program:*

$$\Pi_9 \quad = \quad \left\{ \begin{array}{ll} r_1: & a \leftarrow b \\ r_2: & b \leftarrow a \\ r_3: & b \leftarrow c \\ r_4: & c \leftarrow not\ d \\ r_5: & d \leftarrow not\ c \end{array} \right\}$$

Let us further investigate the following two non-contradictory complete branches:

Π_9	
$\boldsymbol{T}d$	(*Cut*$[atom(\Pi_9)]$)
$\boldsymbol{F}c$	$(N\uparrow, N\downarrow, U\uparrow, U\downarrow)$
$\boldsymbol{F}a$	$(U\uparrow)$
$\boldsymbol{F}b$	$(I\downarrow, N\uparrow, U\uparrow)$

Π_9	
$\boldsymbol{T}a$	(*Cut*$[atom(\Pi_9)]$)
$\boldsymbol{T}b$	$(I\uparrow, N\downarrow, U\downarrow)$
$\boldsymbol{T}c$	$(U\downarrow)$
$\boldsymbol{F}d$	$(N\uparrow, N\downarrow, U\uparrow, U\downarrow)$

We have chosen these branches for illustrating the application of the unfounded set rule ($U\uparrow$*) and the well-founded set rule (*$U\downarrow$*), respectively. (Along branches, we indicate in parentheses all possible rule applications leading to the same result.) We*

[12] The uniqueness of branches stated in the second item of Theorem 3.9 is trivial for unary programs. It becomes more interesting below when composite language constructs are assigned in addition to atoms.

first inspect the deduction of $\boldsymbol{F}a$ by $U\uparrow$ in the left branch. Taking the set $\{a,b\}$ (and its element a) makes us check whether $sup_{\{\boldsymbol{T}d,\boldsymbol{F}c\}}(\Pi_9,\{a,b\},\{a,b\})$ is empty. To this end, we have to inspect all rules that allow for deriving an atom in $\{a,b\}$ (as stipulated via $\overleftarrow{sup}_{\{\boldsymbol{T}d,\boldsymbol{F}c\}}(\alpha,\{a,b\})$). In view of $\boldsymbol{f}c = \boldsymbol{F}c \in \{\boldsymbol{T}d,\boldsymbol{F}c\}$, we have that $r_3 \notin sup_{\{\boldsymbol{T}d,\boldsymbol{F}c\}}(\Pi_9,\{a,b\},\{a,b\})$, so that only r_1 and r_2 require further consideration. Because of $b \in \{a,b\}$ and $a \in \{a,b\}$, respectively, neither of these rules $\alpha \leftarrow \beta$ satisfies $\overrightarrow{sup}_{\{\boldsymbol{T}d,\boldsymbol{F}c\}}(\beta,\{a,b\})$, which leaves us with $sup_{\{\boldsymbol{T}d,\boldsymbol{F}c\}}(\Pi_9,\{a,b\},\{a,b\}) = \emptyset$. After $\boldsymbol{F}a$ has been deduced, it follows that $r_2 \notin sup_{\{\boldsymbol{T}d,\boldsymbol{F}c,\boldsymbol{F}a\}}(\Pi_9,\{b\},S')$ and $r_3 \notin sup_{\{\boldsymbol{T}d,\boldsymbol{F}c,\boldsymbol{F}a\}}(\Pi_9,\{b\},S')$ for $S' \subseteq \{b\}$. Hence, we can deduce $\boldsymbol{F}b$ by $N\uparrow$ or $U\uparrow$, or alternatively by means of $I\downarrow$ in view of rule r_1 and $\boldsymbol{F}a$. On the other hand, the well-founded set inference of $\boldsymbol{T}c$ in the right branch requires a set of atoms, some of whose elements is true, such that only one rule can non-circularly support the set. Taking again $\{a,b\}$, since $\overrightarrow{sup}_{\{\boldsymbol{T}a,\boldsymbol{T}b\}}(b,\{a,b\})$ and $\overrightarrow{sup}_{\{\boldsymbol{T}a,\boldsymbol{T}b\}}(a,\{a,b\})$ do not hold, we get $sup_{\{\boldsymbol{T}a,\boldsymbol{T}b\}}(\Pi_9,\{a,b\},\{a,b\}) = \{b \leftarrow c\}$. The membership of r_3 is justified by $\boldsymbol{f}c = \boldsymbol{F}c \notin \{\boldsymbol{T}a,\boldsymbol{T}b\}$ along with the fact that $\overleftarrow{sup}_{\{\boldsymbol{T}a,\boldsymbol{T}b\}}(b,\{a,b\})$ and $\overrightarrow{sup}_{\{\boldsymbol{T}a,\boldsymbol{T}b\}}(c,\{a,b\})$ hold. Since r_3 is the only rule that can non-circularly support $\{a,b\}$, the presence of $\boldsymbol{T}a$ and $\boldsymbol{T}b$ in the assignment necessitates $\boldsymbol{t}c = \boldsymbol{T}c$, as it is deduced by means of $U\downarrow$. Finally, $\boldsymbol{T}c$ allows us to further deduce $\boldsymbol{F}d$ by $N\downarrow$ or $U\downarrow$ in view of r_4, or alternatively by $N\uparrow$ or $U\uparrow$ in view of r_5.

3.3.3 Conjunctive Bodies

Having settled our constitutive generic framework, we now allow rule bodies to contain conjunctions. While rule bodies are often considered to be conjunctions (as in Section 3.1), we here take a slightly different perspective in viewing conjunctions as (simple) Boolean aggregates, which like atoms can be preceded by not. This gives us some first insights into the treatment of more sophisticated aggregates, such as cardinality constraints to be dealt with afterwards.

A *conjunction* over a denumerable alphabet $\mathcal{P}$ is an expression of the form $\{l_1,\ldots,l_n\}$, where l_i is an atomic literal for $1 \leq i \leq n$. We denote by $conj(\mathcal{P})$ the set of all conjunctions that can be constructed from atoms in $\mathcal{P}$. A rule $\alpha \leftarrow \beta$ such that α is an atomic literal and β is an atomic literal or a possibly negated conjunction of atomic literals is a *conjunctive rule*. A logic program is a *conjunctive program* if it consists of conjunctive rules. For defining the semantics of conjunctive programs, we add the following case to translation $\tau[\pi]$ in (3.6):

$$\tau[\pi] \;=\; \textstyle\bigwedge_{l\in\pi}\tau[l] \qquad \text{if } \pi \in conj(\mathcal{P})$$

For accommodating conjunctions within the generic tableau rules in Figure 3.4, we further extend the previous concepts in (3.8)–(3.13) in a straightforward way:

$$\begin{array}{lcl} \overrightarrow{sup}_{\mathbf{A}}(\{l_1,\ldots,l_n\},S') & & \text{if } \overrightarrow{sup}_{\mathbf{A}}(l,S') \text{ for every } l \in \{l_1,\ldots,l_n\} \\ max_{\mathbf{A}}(\{l_1,\ldots,l_n\},S') & = & \bigcup_{l\in\{l_1,\ldots,l_n\}} max_{\mathbf{A}}(l,S') \end{array}$$

Note that $max_{\mathbf{A}}(\{l_1,\ldots,l_n\},S')$ is still empty since $max_{\mathbf{A}}(l,S') = \emptyset$ for every atomic literal $l \in \{l_1,\ldots,l_n\}$. Thus, it has no effect yet, but this changes when adding cardinality constraints.

For a conjunctive program Π, we fix the domain of assignments $\mathbf{A}$ as well as the cut objects used by *Cut*$[\Gamma]$ to $dom(\mathbf{A}) = \Gamma = atom(\Pi) \cup conj(\Pi)$, where $conj(\Pi)$ is the set

$$\frac{\boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n}{\boldsymbol{T}\{l_1, \ldots, l_n\}}$$

(h) True Conjunction ($TC\uparrow$)

$$\frac{\boldsymbol{F}\{l_1, \ldots, l_{i-1}, l_i, l_{i+1}, \ldots, l_n\} \quad \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_{i-1}, \boldsymbol{t}l_{i+1}, \ldots, \boldsymbol{t}l_n}{\boldsymbol{f}l_i}$$

(i) Falsify Conjunction ($TC\downarrow$)

$$\frac{\boldsymbol{f}l_i}{\boldsymbol{F}\{l_1, \ldots, l_i, \ldots, l_n\}}$$

(j) False Conjunction ($FC\uparrow$)

$$\frac{\boldsymbol{T}\{l_1, \ldots, l_n\}}{\boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n}$$

(k) Justify Conjunction ($FC\downarrow$)

Figure 3.5: Tableau rules for conjunctions.

of conjunctions occurring in Π. The additional tableau rules for handling conjunctions are shown in Figure 3.5. Their purpose is to ensure that $\boldsymbol{T}\{l_1, \ldots, l_n\} \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap \mathcal{P} \models (\tau[l_1] \wedge \cdots \wedge \tau[l_n])$ holds for total assignments $\mathbf{A}$. By augmenting our generic calculus with the tableau rules in Figure 3.5, Theorem 3.9 extends to conjunctive programs.

Theorem 3.10. *Let Π be a conjunctive program.*

Then, we have that Statement 1, 2, and 3 given in Theorem 3.9 hold for the tableau calculus consisting of the tableau rules (a)–(k).

It is interesting to note that conjunctive programs extend normal programs by admitting negative literals in heads of rules and rule bodies to be (default) negated conjunctions. This implies that normal programs are conjunctive and can thus be treated using the tableau rules *(a)–(k)*.[13] The naturally arising question is how generic tableau rules relate to the ones in Figure 3.1 on Page 19, which were specialized to normal programs. To answer it, Table 3.1 shows the inherent correspondences between both kinds of tableau rules. For instance, the tableau rule *FTA*, allowing for making derivable atoms true, achieves the same effect as the generic implication rule $I\uparrow$. In fact, we obtain the following correspondence result for normal programs.

Proposition 3.11. *Let Π be a normal program, $\mathbf{A}$ an assignment, and F, G any pair of a basic tableau rule F and a generic tableau rule G belonging to the same line in Table 3.1.*

Then, we have that

1. $D_{\{F\}}(\Pi, \mathbf{A}) = D_{\{G\}}(\Pi, \mathbf{A})$ *if* $F \notin \{$*BTA*, *WFJ*$[2^{atom(\Pi)}]\}$;
2. $D_{\{\textit{BTA}\}}(\Pi, \mathbf{A}) \supseteq D_{\{N\downarrow\}}(\Pi, \mathbf{A})$ *and, if* $D_{\{\textit{BTA}\}}(\Pi, \mathbf{A}) \neq D_{\{N\downarrow\}}(\Pi, \mathbf{A})$, *then* $\mathbf{A} \cup D_{\{N\uparrow\}}(\Pi, \mathbf{A})$ *is contradictory;*
3. $D_{\{\textit{WFJ}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) \supseteq D_{\{U\downarrow\}}(\Pi, \mathbf{A})$ *and, if* $\boldsymbol{T}B \in D_{\{\textit{WFJ}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) \setminus D_{\{U\downarrow\}}(\Pi, \mathbf{A})$, *then* $\mathbf{A} \cup D_{\{U\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{F}B\})$ *is contradictory.*

This shows that similar entries are deducible by either kind of tableau rules. A technical difference, though, is that, if *BTA* is applicable because of some $p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ such that $body_\Pi(p) \subseteq \mathbf{A}^{\boldsymbol{F}}$, then $N\downarrow$ is not applicable to p since $sup_{\mathbf{A}}(\Pi, \{p\}, \emptyset) = \emptyset$.

[13] Answer sets of $\tau[\Pi]$ match answer sets (as introduced in Section 2.1) of a normal program Π (cf. [162]).

Basic Tableau Rule			Generic Tableau Rule		
(c)	Forward True Atom	*FTA*	*(a)*	Implication	$I\uparrow$
(d)	Backward False Atom	*BFA*	*(b)*	Contraposition	$I\downarrow$
(g)	Forward False Atom	*FFA*	*(c)*	Negation	$N\uparrow$
(h)	Backward True Atom	*BTA*	*(d)*	Support	$N\downarrow$
(i)	Well-Founded Negation	*WFN*$[2^{atom(\Pi)}]$	*(e)*	Unfounded Set	$U\uparrow$
(j)	Well-Founded Justification	*WFJ*$[2^{atom(\Pi)}]$	*(f)*	Well-Founded Set	$U\downarrow$
(a)	Forward True Body	*FTB*	*(h)*	True Conjunction	$TC\uparrow$
(b)	Backward False Body	*BFB*	*(i)*	Falsify Conjunction	$TC\downarrow$
(e)	Forward False Body	*FFB*	*(j)*	False Conjunction	$FC\uparrow$
(f)	Backward True Body	*BTB*	*(k)*	Justify Conjunction	$FC\downarrow$

Table 3.1: Correspondences between basic and generic tableau rules (for normal programs).

In such a case, a contradictory assignment is obtained by applying $N\uparrow$. Furthermore, if $\boldsymbol{T}B$ is deduced by *WFJ*$[2^{atom(\Pi)}]$ in view of some $S \subseteq atom(\Pi)$ such that $\mathbf{A}^{\boldsymbol{T}} \cap S \neq \emptyset$ and $EB_\Pi(S) \setminus \mathbf{A}^{\boldsymbol{F}} \subseteq \{B\}$, there may be none or multiple rules $p \leftarrow B$ in Π for which $p \in S$, so that $|sup_{\mathbf{A}}(\Pi, S, S)| = 1$ is not guaranteed. In such a case, applying $U\uparrow$ w.r.t. $\mathbf{A} \cup \{\boldsymbol{F}B\}$ yields a contradiction. That is, an entry $\boldsymbol{T}B$ deducible by *WFJ*$[2^{atom(\Pi)}]$ can also be obtained with the generic calculus by means of cutting (on B or its body literals) and closing branches with $\boldsymbol{F}B$ via $U\uparrow$, which yields $\boldsymbol{T}B$ in the single remaining branch. Hence, differences between *BTA* and $N\downarrow$ as well as *WFJ*$[2^{atom(\Pi)}]$ and $U\downarrow$ are merely technical, but not fundamental, discrepancies.

We further define the *generic image* of a basic calculus $\mathcal{T}$ as the generic calculus $\mathcal{T}'$ containing the *Cut* rules of $\mathcal{T}$ and the generic tableau rules associated with basic tableau rules in $\mathcal{T}$ according to Table 3.1. Then, we obtain the following from Proposition 3.11.

Proposition 3.12. *Let Π be a normal program, $\mathbf{A}$ an assignment, $\mathcal{T}$ a tableau calculus containing any subset of the tableau rules in Figure 3.1 for $\Omega = 2^{atom(\Pi)}$, and $\mathcal{T}'$ the generic image of $\mathcal{T}$.*

If FFA $\in \mathcal{T}$ or BTA $\notin \mathcal{T}$ and if WFJ$[\Omega] \in \mathcal{T}$ implies that $\{$FTB, FFB, WFN$[\Omega]$, Cut$[\Gamma]\} \subseteq \mathcal{T}$ for $\Gamma \subseteq atom(\Pi) \cup body(\Pi)$ such that $atom(\Pi) \subseteq \Gamma$ or $body(\Pi) \subseteq \Gamma$, then we have that the following holds:

1. *For every complete tableau of $\mathcal{T}$ for Π and $\mathbf{A}$ with n branches, there is a complete tableau of $\mathcal{T}'$ for Π and $\mathbf{A}$ with the same non-contradictory branches and at most $(\max\{|atom(\Pi)|, |body(\Pi)|\} + 1) * n$ branches overall.*
2. *Every tableau of $\mathcal{T}'$ for Π and $\mathbf{A}$ is a tableau of $\mathcal{T}$ for Π and $\mathbf{A}$.*

While every tableau of the generic image $\mathcal{T}'$ is likewise a tableau of $\mathcal{T}$, in view of Proposition 3.11, deductions by *BTA* or *WFJ*$[\Omega]$ may not be obtainable with $N\downarrow$ or $U\downarrow$, respectively. Such deductions can still be simulated by means of other generic tableau rules, possibly introducing a polynomial number of additional branches, for which the approximation given in the first item of Proposition 3.12 provides an upper limit.

Example 3.18. *To illustrate the correspondence between basic and generic tableau rules, reconsider the normal program Π_1 from Example 2.1. The tableau of $\mathcal{T}_{smodels}$ for Π_1 and the empty assignment shown in Figure 3.2 on Page 20 can also be generated by the generic image of $\mathcal{T}_{smodels}$, consisting of the generic tableau rules (a)–(e), (h)–(k), and*

$$\left\{\begin{array}{ll} r_1: & a \leftarrow \\ r_2: & c \leftarrow not\ b, not\ d \\ r_3: & d \leftarrow a, not\ c \end{array}\right\}$$

		$\boldsymbol{T}\emptyset$			$(TC\uparrow)$
		$\boldsymbol{T}a$			$(I\uparrow)$
		$\boldsymbol{F}b$			$(N\uparrow)$
$\boldsymbol{T}c$			$\boldsymbol{F}c$		$(\textbf{\textit{Cut}}[atom(\Pi_1)])$
$\boldsymbol{T}\{not\ b, not\ d\}$	$(N\downarrow)$		$\boldsymbol{F}\{not\ b, not\ d\}$	$(I\downarrow)$	
$\boldsymbol{F}d$	$(FC\downarrow)$		$\boldsymbol{T}d$	$(TC\downarrow)$	
$\boldsymbol{F}\{a, not\ c\}$	$(FC\uparrow)$		$\boldsymbol{T}\{a, not\ c\}$	$(TC\uparrow)$	

Figure 3.6: Complete tableau of the generic image of $\mathcal{T}_{smodels}$ for Π_1 and the empty assignment.

Cut$[atom(\Pi)]$. *(Note that, like* $\mathcal{T}_{smodels}$*, its generic image restricts cut objects to atoms, and it does not include* $U\downarrow$ *because* $\mathcal{T}_{smodels}$ *does not contain its associated basic tableau rule WFJ*$[2^{atom(\Pi)}]$*.) The generic tableau resembling the one in Figure 3.2 is shown in Figure 3.6.*[14] *It is obtained by replacing the references to applied tableau rules adequately based on the mapping in Table 3.1. Converse replacements of generic by basic tableau rules are also possible, provided that a logic program at hand is normal.*

3.3.4 Cardinality Constraints

We further extend our generic tableau framework to logic programs including cardinality constraints [209]. The expressiveness of such programs, in terms of compact modeling, goes well beyond the one of normal (and conjunctive) programs. Hence, the integration of cardinality constraints sheds light on how to extend our generic framework to accommodate (sophisticated) aggregates.

A *cardinality constraint* over a denumerable alphabet $\mathcal{P}$ is an expression of the form $j\{l_1, \ldots, l_n\}k$, where l_i is an atomic literal for $1 \leq i \leq n$ and j, k are integers such that $0 \leq j \leq k \leq n$. We denote by $card(\mathcal{P})$ the set of all cardinality constraints that can be constructed from atoms in $\mathcal{P}$. For $v \in \mathcal{P} \cup card(\mathcal{P})$, we say that v and $not\ v$ are *cardinality literals*. A rule $\alpha \leftarrow \beta$ such that α is a cardinality literal and β is a cardinality literal or a possibly negated conjunction of cardinality literals is a *cardinality rule*. A logic program is a *cardinality program* if it consists of cardinality rules.

For cardinality constraints in heads of rules, we adopt the approach of [209, 72, 171] and interpret them as "choice constructs," meaning that atoms are not minimized within such cardinality constraints. To reflect the "lack of minimization," cardinality constraints in heads of rules necessitate an extended translation of cardinality programs to propositional theories.[15] Hence, we let $atom(j\{l_1, \ldots, l_n\}k) = \{l_1, \ldots, l_n\} \cap \mathcal{P}$ and replace the definition of $\tau[\Pi]$ in (3.5) by the following translation:

[14] In Figure 3.6 and in the sequel, we skip set notation for conjunctions within bodies of rules, like $\emptyset$, $\{not\ b, not\ d\}$, and $\{a, not\ c\}$ in the bodies of rules in Π_1.

[15] Interpreting aggregates in heads of rules as "choice constructs" avoids an increase of computational complexity by one level in the polynomial time hierarchy. If derivable atoms were to be minimized, it would be straightforward to embed disjunctive programs (considered in Section 3.3.5) into cardinality programs.

$$\begin{array}{rcl} \tau[\Pi] & = & \{\tau[\beta] \rightarrow \tau[\alpha] \mid (\alpha \leftarrow \beta) \in \Pi, \alpha \notin \mathit{card}(\mathcal{P})\} \\ & \cup & \{\tau[\beta] \rightarrow \big(\tau[\alpha] \wedge \bigwedge_{p \in atom(\alpha)}(p \vee \neg p)\big) \mid (\alpha \leftarrow \beta) \in \Pi, \alpha \in \mathit{card}(\mathcal{P})\} \end{array} \tag{3.14}$$

Note that conjuncts $p \vee \neg p$ are tautological and thus neutral as regards the (classical) models of $\tau[\Pi]$. Given an interpretation X, they however justify the truth of all $p \in atom(\alpha) \cap X$ in $(\tau[\Pi])^X$, in which $\neg p$ is replaced by $\bot$. We further add another case to translation $\tau[\pi]$ in (3.6), which arises from the general aggregate semantics in [69]:

$$\begin{array}{rcl} \tau[\pi] & = & \bigwedge_{L \subseteq \{l_1,\ldots,l_n\}, |L|<j \text{ or } k<|L|} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L} \tau[l])\big) \\ & & \text{if } \pi = j\{l_1, \ldots, l_n\}k \in \mathit{card}(\mathcal{P}) \end{array}$$

Since $\tau[j\{l_1, \ldots, l_n\}k]$ inspects individual subsets of $\{l_1, \ldots, l_n\}$ (of cardinality below j or beyond k), its size is, in general, exponential in n. Albeit the use of auxiliary atoms admits a polynomial translation of cardinality constraints to normal rules as, e.g., described in [209], compilation approaches incur a significant blow-up in space. Our proof-theoretic characterizations given below apply directly to cardinality constraints and thus avoid any such blow-up.

For a cardinality program Π, we fix the domain of assignments $\mathbf{A}$ as well as the cut objects used by *Cut*$[\Gamma]$ to $dom(\mathbf{A}) = \Gamma = atom(\Pi) \cup \mathit{conj}(\Pi) \cup \mathit{card}(\Pi)$, where $\mathit{card}(\Pi)$ is the set of cardinality constraints occurring in Π. Figure 3.7 shows the tableau rules augmenting those in Figure 3.4 and 3.5 to additionally handle cardinality constraints. These rules match truth values of atoms occurring in a cardinality constraint to the one assigned to the constraint as a whole, in order to ensure that $\boldsymbol{T}j\{l_1, \ldots, l_n\}k \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap \mathcal{P} \models \tau[j\{l_1, \ldots, l_n\}k]$ holds for total assignments $\mathbf{A}$.

Example 3.19. *Consider the following cardinality constraint:*

$$\gamma \quad = \quad 2\{a, b, c, \mathit{not}\ d, \mathit{not}\ e\}3$$

Cardinality constraint γ includes $n = 5$ literals, lower bound $j = 2$, and upper bound $k = 3$. For an assignment $\mathbf{A}$, tableau rule $TLU{\uparrow}$ allows for deducing $\boldsymbol{T}\gamma$ if at least $j = 2$ literals l of γ hold (i.e., $\boldsymbol{t}l \in \mathbf{A}$) and at least $n - k = 2$ literals l of γ are false (i.e., $\boldsymbol{f}l \in \mathbf{A}$). This applies, for instance, to the assignment $\mathbf{A}_1 = \{\boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{T}d, \boldsymbol{F}e\}$; hence, $\boldsymbol{T}\gamma$ can be deduced by $TLU{\uparrow}$. Indeed, the lower and the upper bound of γ are respected in every non-contradictory assignment that extends $\mathbf{A}_1$, no matter whether $\boldsymbol{T}c$ or $\boldsymbol{F}c$ is additionally included. Tableau rules $TLU{\downarrow}$ and $TLU{\downarrow}$ are the contrapositives of $TLU{\uparrow}$, ensuring that either the lower or the upper bound of γ is violated if $\boldsymbol{F}\gamma$ belongs to an assignment. For instance, $\boldsymbol{F}c$ and $\boldsymbol{T}e$ can be deduced by $TLU{\downarrow}$ w.r.t. the assignment $\mathbf{A}_2 = \{\boldsymbol{F}\gamma, \boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{T}d\}$. Observe that the upper bound $k = 3$ cannot be violated in non-contradictory extensions of $\mathbf{A}_2$, since $n - k = 2$ literals of γ are already false w.r.t. $\mathbf{A}_2$, containing $\boldsymbol{F}b$ and $\boldsymbol{T}d$. On the other hand, $TLU{\downarrow}$ allows for deducing $\boldsymbol{T}c$ and $\boldsymbol{F}e$ w.r.t. $\{\boldsymbol{F}\gamma, \boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{F}d\}$ in order to violate the upper bound $k = 3$; the lower bound $j = 2$ cannot be violated because of $\boldsymbol{T}a$ and $\boldsymbol{F}d$. The remaining four tableau rules in Figure 3.7 allow for deducing $\boldsymbol{F}\gamma$ if its lower ($FL{\uparrow}$) or upper ($FU{\uparrow}$) bound is violated, or for making sure that the lower ($FL{\downarrow}$) and the upper ($FU{\downarrow}$) bound are respected if $\boldsymbol{T}\gamma$ belongs to an assignment. For instance, $\boldsymbol{F}\gamma$ can be deduced by $FL{\uparrow}$ w.r.t. the assignment $\{\boldsymbol{F}b, \boldsymbol{F}c, \boldsymbol{T}d, \boldsymbol{T}e\}$, and by $FU{\uparrow}$ w.r.t. $\{\boldsymbol{T}b, \boldsymbol{T}c, \boldsymbol{F}d, \boldsymbol{F}e\}$. Conversely, $FL{\downarrow}$ allows for deducing $\boldsymbol{T}a$ and $\boldsymbol{F}e$ w.r.t. $\{\boldsymbol{T}\gamma, \boldsymbol{F}b, \boldsymbol{F}c, \boldsymbol{T}d\}$, and $FU{\downarrow}$ allows for deducing $\boldsymbol{F}a$ and $\boldsymbol{T}e$ w.r.t. $\{\boldsymbol{T}\gamma, \boldsymbol{T}b, \boldsymbol{T}c, \boldsymbol{F}d\}$.

$$\frac{\boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_j, \boldsymbol{f}l_{k+1}, \ldots, \boldsymbol{f}l_n}{\boldsymbol{T}j\{l_1, \ldots, l_j, \ldots, l_{k+1}, \ldots, l_n\}k}$$

(l) True Bounds ($TLU\uparrow$)

$$\frac{\begin{array}{c}\boldsymbol{F}j\{l_1, \ldots, l_{j-1}, l_j, \ldots, l_k, l_{k+1}, \ldots, l_n\}k \\ \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_{j-1}, \boldsymbol{f}l_{k+1}, \ldots, \boldsymbol{f}l_n\end{array}}{\boldsymbol{f}l_j, \ldots, \boldsymbol{f}l_k}$$

(m) Falsify Lower Bound ($TLU\downarrow$)

$$\frac{\begin{array}{c}\boldsymbol{F}j\{l_1, \ldots, l_j, l_{j+1}, \ldots, l_{k+1}, l_{k+2}, \ldots, l_n\}k \\ \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_j, \boldsymbol{f}l_{k+2}, \ldots, \boldsymbol{f}l_n\end{array}}{\boldsymbol{t}l_{j+1}, \ldots, \boldsymbol{t}l_{k+1}}$$

(n) Falsify Upper Bound ($TLU\downarrow$)

$$\frac{\boldsymbol{f}l_j, \ldots, \boldsymbol{f}l_n}{\boldsymbol{F}j\{l_1, \ldots, l_j, \ldots, l_n\}k}$$

(o) False Lower Bound ($FL\uparrow$)

$$\frac{\begin{array}{c}\boldsymbol{T}j\{l_1, \ldots, l_j, l_{j+1}, \ldots, l_n\}k \\ \boldsymbol{f}l_{j+1}, \ldots, \boldsymbol{f}l_n\end{array}}{\boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_j}$$

(p) Justify Lower Bound ($FL\downarrow$)

$$\frac{\boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_{k+1}}{\boldsymbol{F}j\{l_1, \ldots, l_{k+1}, \ldots, l_n\}k}$$

(q) False Upper Bound ($FU\uparrow$)

$$\frac{\begin{array}{c}\boldsymbol{T}j\{l_1, \ldots, l_k, l_{k+1}, \ldots, l_n\}k \\ \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_k\end{array}}{\boldsymbol{f}l_{k+1}, \ldots, \boldsymbol{f}l_n}$$

(r) Justify Upper Bound ($FU\downarrow$)

Figure 3.7: Tableau rules for cardinality constraints.

To integrate cardinality constraints into the generic setting of the tableau rules in Figure 3.4, we also need to extend the concepts in (3.8)–(3.13):

$$\begin{array}{lcl}
\overleftarrow{sup}_{\mathbf{A}}(j\{l_1,\ldots,l_n\}k,S) & \text{if} & \{l_1,\ldots,l_n\}\cap S\neq\emptyset \text{ and} \\
 & & \quad |\{l\in\{l_1,\ldots,l_n\}\setminus S \mid \boldsymbol{t}l\in\mathbf{A}\}|<k \\
\overrightarrow{sup}_{\mathbf{A}}(j\{l_1,\ldots,l_n\}k,S') & \text{if} & |\{l\in\{l_1,\ldots,l_n\}\setminus S' \mid \boldsymbol{f}l\notin\mathbf{A}\}|\geq j \\
min_{\mathbf{A}}(j\{l_1,\ldots,l_n\}k,S) & = & \left\{\begin{array}{l}\{\boldsymbol{f}l \mid l\in\{l_1,\ldots,l_n\}\setminus S,\ \boldsymbol{t}l\notin\mathbf{A}\} \\ \quad \text{if } |\{l\in\{l_1,\ldots,l_n\}\setminus S \mid \boldsymbol{t}l\in\mathbf{A}\}|=k-1 \\ \emptyset \text{ if } |\{l\in\{l_1,\ldots,l_n\}\setminus S \mid \boldsymbol{t}l\in\mathbf{A}\}|\neq k-1\end{array}\right. \\
max_{\mathbf{A}}(j\{l_1,\ldots,l_n\}k,S') & = & \left\{\begin{array}{l}\{\boldsymbol{t}l \mid l\in\{l_1,\ldots,l_n\}\setminus S',\ \boldsymbol{f}l\notin\mathbf{A}\} \\ \quad \text{if } |\{l\in\{l_1,\ldots,l_n\}\setminus S' \mid \boldsymbol{f}l\notin\mathbf{A}\}|=j \\ \emptyset \text{ if } |\{l\in\{l_1,\ldots,l_n\}\setminus S' \mid \boldsymbol{f}l\notin\mathbf{A}\}|\neq j\end{array}\right.
\end{array}$$

Recall that $\overleftarrow{sup}_{\mathbf{A}}(\alpha,S)$ is used to determine whether a rule with head α can provide support for the atoms in S. If $\alpha=j\{l_1,\ldots,l_n\}k$, then some atom of S must belong to $\{l_1,\ldots,l_n\}$. Furthermore, if $\{l_1,\ldots,l_n\}\setminus S$ already contains k (or more) literals that hold w.r.t. $\mathbf{A}$, then the addition of $\boldsymbol{T}p$ to $\mathbf{A}$ for $p\in\{l_1,\ldots,l_n\}\cap S$ would violate the upper bound k, so that the corresponding rule $\alpha\leftarrow\beta$ cannot support S. This also explains the false literals in $min_{\mathbf{A}}(j\{l_1,\ldots,l_n\}k,S)$ that can be deduced if $k-1$ literals of $\{l_1,\ldots,l_n\}\setminus S$ hold already. In addition, $\overrightarrow{sup}_{\mathbf{A}}(\beta,S')$ is used to verify whether a support via β is external to S'. If, for a rule $\alpha\leftarrow\beta$, either $\beta=j\{l_1,\ldots,l_n\}k$ or β is a conjunction such that $j\{l_1,\ldots,l_n\}k\in\beta$, then there must be enough non-false literals w.r.t. $\mathbf{A}$ in $\{l_1,\ldots,l_n\}\setminus S'$ to achieve the lower bound j. If the number of such literals is exactly j, then all of them must hold for providing a support that is external to S'. This is expressed by $max_{\mathbf{A}}(j\{l_1,\ldots,l_n\}k,S')$.

Example 3.20. *Consider the following cardinality program:*

$$\Pi_{10} \;=\; \left\{\begin{array}{ll} r_1: & 0\{c,d,e\}3\leftarrow \\ r_2: & \quad 1\{a,b\}2\leftarrow c,d \\ r_3: & \quad 0\{a,d\}1\leftarrow 1\{b,not\ e\}2 \\ r_4: & \quad 1\{b,d\}2\leftarrow 1\{a,c\}2 \\ r_5: & \quad 1\{a,d\}2\leftarrow b \end{array}\right\}$$

Let $\mathbf{A}=\{\boldsymbol{T}a,\boldsymbol{F}c,\boldsymbol{F}\{c,d\}\}$, and note that tableau rule $U{\downarrow}$ (or $N{\downarrow}$) does not apply to the set $\{a\}$ since $sup_{\mathbf{A}}(\Pi_{10},\{a\},\{a\})=\{r_3,r_5\}$. We further consider the set $\{a,b\}$. Given that $\{c,d,e\}\cap\{a,b\}=\emptyset$ and $\boldsymbol{F}\{c,d\}\in\mathbf{A}$, we have that $r_1\notin sup_{\mathbf{A}}(\Pi_{10},\{a,b\},\{a,b\})$ and $r_2\notin sup_{\mathbf{A}}(\Pi_{10},\{a,b\},\{a,b\})$. Regarding the body of r_5, $b\in\{a,b\}$ is the reason for $\overrightarrow{sup}_{\mathbf{A}}(b,\{a,b\})$ not to hold. For the body of r_4, we have that $\{a,c\}\setminus\{a,b\}=\{c\}$ and $\boldsymbol{f}c=\boldsymbol{F}c\in\mathbf{A}$, so that there are no non-false literals in $\{a,c\}\setminus\{a,b\}$. That is, the lower bound 1 of $1\{a,c\}2$ cannot be achieved independently of $\{a,b\}$, and thus $\overrightarrow{sup}_{\mathbf{A}}(1\{a,c\}2,\{a,b\})$ does not hold. We have now established that only r_3 is potentially contained in $sup_{\mathbf{A}}(\Pi_{10},\{a,b\},\{a,b\})$. As not $e\in\{b,not\ e\}$ is a non-false literal not belonging to $\{a,b\}$, $\overrightarrow{sup}_{\mathbf{A}}(1\{b,not\ e\}2,\{a,b\})$ holds. In addition, $\overleftarrow{sup}_{\mathbf{A}}(0\{a,d\}1,\{a,b\})$ holds because $\{a,d\}\cap\{a,b\}\neq\emptyset$ and $\boldsymbol{t}d=\boldsymbol{T}d\notin\mathbf{A}$. This shows that $sup_{\mathbf{A}}(\Pi_{10},\{a,b\},\{a,b\})=\{r_3\}$. As entries deducible by $U{\downarrow}$, we obtain $\boldsymbol{t}1\{b,not\ e\}2=\boldsymbol{T}1\{b,not\ e\}2$, $min_{\mathbf{A}}(0\{a,d\}1,\{a,b\})=\{\boldsymbol{f}d\}=\{\boldsymbol{F}d\}$, and $max_{\mathbf{A}}(1\{b,not\ e\}2,\{a,b\})=\{\boldsymbol{t}not\ e\}=\{\boldsymbol{F}e\}$. In fact, if $\boldsymbol{T}d$ or $\boldsymbol{T}e$ had been contained in $\mathbf{A}$, we would have obtained $sup_{\mathbf{A}}(\Pi_{10},\{a,b\},\{a,b\})=\emptyset$, so that deducing $\boldsymbol{F}a$

by $U\uparrow$ would have led to a contradiction. Also note that the only answer set of Π_{10} compatible with both $\boldsymbol{T}a$ and $\boldsymbol{F}c$, $\{a, b\}$, does not include d and e.

In view of the generalizations of $\overleftarrow{sup}$, $\overrightarrow{sup}$, min, and max provided above, Theorem 3.9 extends to cardinality programs.

Theorem 3.13. *Let Π be a cardinality program.*

Then, we have that Statement 1, 2, and 3 given in Theorem 3.9 hold for the tableau calculus consisting of the tableau rules (a)–(r).

On the example of cardinality constraints, we have demonstrated the full granularity of our generic tableau rules in Figure 3.4, making use of the extended definitions of $\overleftarrow{sup}$, $\overrightarrow{sup}$, min, and max. Importantly, we have extended the domain of assignments (and cut objects) to include cardinality constraints. As a matter of fact, this admits the addition of the tableau rules in Figure 3.7, matching the truth value of a cardinality constraint to those of its constituents, without any modification of tableau rules dealing with other language constructs, like the ones for conjunctions in Figure 3.5. The same methodology could be used to deal with logic programs further including *smodels*' weight constraints [209] or *dlv*'s aggregates [67]. Notably, the approach of *clasp* to incorporate extended language constructs [84] is closely related: *clasp* extends assignments and unit propagation to composite language constructs for keeping track of reasons for inferences, which can be extracted from tableau rules by using the methodology described in Section 3.2.3.

3.3.5 Disjunctive Heads

The extension of normal programs' syntax and semantics by allowing heads of rules to be (proper) disjunctions of atoms is one of the earliest generalizations of answer set semantics [120]. Due to admitting the (unrestricted) use of disjunction, the inherent computational complexity of important reasoning tasks increases from the first to the second level of the polynomial time hierarchy [57, 158]. As we show in the following, it is nonetheless possible to extend our generic tableau framework by allowing (proper) disjunctions in heads of rules, without imposing any restrictions on computational complexity.

A *disjunction* over a denumerable alphabet $\mathcal{P}$ is an expression of the form $\{l_1; \ldots; l_n\}$, where l_i is an atomic literal for $1 \leq i \leq n$. We denote by $disj(\mathcal{P})$ the set of all disjunctions that can be constructed from atoms in $\mathcal{P}$. For $v \in \mathcal{P} \cup card(\mathcal{P}) \cup disj(\mathcal{P})$, v and $not\ v$ are *disjunctive literals*. A rule $\alpha \leftarrow \beta$ such that α is a disjunctive literal and β is a cardinality literal or a possibly negated conjunction of cardinality literals is a *disjunctive rule*. A logic program is a *disjunctive program* if it consists of disjunctive rules.

In contrast to cardinality constraints serving as "choice constructs," the common semantics for disjunctions relies on the minimization of derivable atoms. Hence, we adhere to the definition of $\tau[\Pi]$ in (3.14) and just add another case to $\tau[\pi]$ in (3.6):

$$\tau[\pi] \quad = \quad \bigvee_{l \in \{l_1, \ldots, l_n\}} \tau[l] \qquad \text{if } \pi = \{l_1; \ldots; l_n\} \in disj(\mathcal{P})$$

We further extend the concepts in (3.8)–(3.13) to disjunctive heads:

$$\begin{aligned} \overleftarrow{sup}_{\mathbf{A}}(\{l_1; \ldots; l_n\}, S) \quad & \text{if} \quad \{l_1, \ldots, l_n\} \cap S \neq \emptyset \text{ and} \\ & \qquad \{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\} = \emptyset \\ min_{\mathbf{A}}(\{l_1; \ldots; l_n\}, S) \quad & = \quad \{\boldsymbol{f}l \mid l \in \{l_1, \ldots, l_n\} \setminus S\} \end{aligned}$$

$$\frac{\boldsymbol{t}l_i}{\boldsymbol{T}\{l_1;\ldots;l_i;\ldots;l_n\}}$$

(s) True Disjunction ($TD\uparrow$)

$$\frac{\boldsymbol{F}\{l_1;\ldots;l_n\}}{\boldsymbol{f}l_1,\ldots,\boldsymbol{f}l_n}$$

(t) Falsify Disjunction ($TD\downarrow$)

$$\frac{\boldsymbol{f}l_1,\ldots,\boldsymbol{f}l_n}{\boldsymbol{F}\{l_1;\ldots;l_n\}}$$

(u) False Disjunction ($FD\uparrow$)

$$\frac{\boldsymbol{T}\{l_1;\ldots;l_{i-1};l_i;l_{i+1};\ldots;l_n\}\quad \boldsymbol{f}l_1,\ldots,\boldsymbol{f}l_{i-1},\boldsymbol{f}l_{i+1},\ldots,\boldsymbol{f}l_n}{\boldsymbol{t}l_i}$$

(v) Justify Disjunction ($FD\downarrow$)

Figure 3.8: Tableau rules for disjunctions.

Observe that the notion of support, $\overleftarrow{sup}_{\mathbf{A}}(\{l_1;\ldots;l_n\},S)$, is closely related to, yet simpler than, the corresponding concept for cardinality constraints, given that disjunctions do not possess an upper bound k. Rather, support requires all literals of $\{l_1,\ldots,l_n\}\setminus S$ to be false; this condition can be established by means of $min_{\mathbf{A}}(\{l_1;\ldots;l_n\},S)$, used by generic tableau rules $N\downarrow$ and $U\downarrow$.

For a disjunctive program Π, we fix the domain of assignments $\mathbf{A}$ as well as the cut objects used by *Cut*$[\Gamma]$ to $dom(\mathbf{A}) = \Gamma = atom(\Pi)\cup conj(\Pi)\cup card(\Pi)\cup disj(\Pi)$, where $disj(\Pi)$ is the set of disjunctions occurring in Π. The additional tableau rules for handling disjunctions are shown in Figure 3.8. Their purpose is to ensure that $\boldsymbol{T}\{l_1;\ldots;l_n\}\in\mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}}\cap\mathcal{P}\models(\tau[l_1]\vee\cdots\vee\tau[l_n])$ holds for total assignments $\mathbf{A}$. The tableau calculus for disjunctive programs is obtained by adding the rules in Figure 3.8 to the ones in Figure 3.4, 3.5, and 3.7. Then, Theorem 3.9 extends to disjunctive programs.

Theorem 3.14. *Let Π be a disjunctive program.*

Then, we have that Statement 1, 2, and 3 given in Theorem 3.9 hold for the tableau calculus consisting of the tableau rules (a)–(v).

As with conjunctive programs that are slightly more general than normal ones, we have a parallel relationship between our and the traditional concept of a disjunctive program [120], given that we admit (default) negation in front of and within a disjunction.[16] However, as the equivalences discussed in [165] show, rules with negative formulas in the head can be rewritten such that occurrences of negation are limited to rule bodies. In our setting, the generic tableau rules in Figure 3.4 tolerate negative literals in heads of rules, so that they can be admitted without difficulties. Given this, the class of disjunctive programs we consider here is expressive enough to represent any nested program [165] (in which heads and bodies of rules are allowed to be formulas).

As regards computational complexity of reasoning tasks, e.g., answer set existence, which can increase due to disjunctions in heads of rules, it does not impose any extra difficulties in specifying tableau rules. Rather, elevated complexity manifests itself in the hardness of verifying the application conditions of tableau rules, namely, the ones of $U\uparrow$ and $U\downarrow$ dealing with unfounded sets. While such conditions can be checked in polynomial time for cardinality programs (cf. [209]), determining a (non-empty) unfounded set is NP-complete for disjunctive programs (cf. [159]). However, it is interesting to note

[16] Unlike [120], we do not consider classical negation, but it could be handled easily by compilation.

that the condition $sup_{\mathbf{A}}(\Pi, S, S) = \emptyset$, checked in the proviso of $U\uparrow$, along with Theorem 3.14 contribute a definition of unfounded sets for disjunctive programs including cardinality constraints. Restricting $\mathbf{A}$ to entries over atoms only also yields a counterpart of GRS-unfounded sets (cf. Definition 2.1 on Page 12) for such disjunctive programs. (It merely requires replacing the support condition $\boldsymbol{f}\beta \notin \mathbf{A}$ in (3.7) by an evaluation of β w.r.t. assigned atoms.) To our knowledge, there is no direct unfounded set definition (not relying on compilation to basic language constructs) for the class of disjunctive programs considered here,[17] yet soundness and completeness results (like Theorem 3.14) in the presence of $U\uparrow$ inherently contribute unfounded set definitions for extended classes of logic programs.

3.4 Proof Complexity

In Section 3.2, we have seen that native ASP solvers largely coincide on their propagation rules and differ primarily in the usage of *Cut*. In this section, we analyze the relative efficiency of tableau calculi w.r.t. different *Cut* rules. We start by taking $\mathcal{T}_{smodels}$, $\mathcal{T}_{nomore}$, and $\mathcal{T}_{nomore++}$ (defined on Page 20) into account, all using the deterministic tableau rules *(a)–(i)* in Figure 3.1 on Page 19 but applying *Cut* to either $atom(\Pi)$, $body(\Pi)$, or both of them. These three calculi are of particular interest as they closely characterize strategies of existing ASP solvers, viz., *smodels* (and *dlv*), *nomore*, and *nomore++*. After in Section 3.4.1 dealing with calculi aiming at normal programs, in Section 3.4.2, we extend our analysis to generic calculi and the impact of *Cut* w.r.t. extended language constructs.

For comparing tableau calculi, we use the concept of *proof complexity* [39] and evaluate the relative efficiency of calculi on unsatisfiable logic programs (having no answer set) in terms of *minimal* refutations. The size of a tableau is determined in the standard way by the number of its nodes (program rules and entries).[18] A tableau calculus $\mathcal{T}$ is *not polynomially simulated* [19, 144] by another calculus $\mathcal{T}'$ if there is an infinite (witnessing) family $\{\Pi^n\}$ of unsatisfiable logic programs such that the asymptotic size of minimal refutations of $\mathcal{T}'$ for Π is exponentially greater than the asymptotic size of minimal refutations of $\mathcal{T}$ for Π. A tableau calculus $\mathcal{T}$ is *exponentially stronger* than a tableau calculus $\mathcal{T}'$ if $\mathcal{T}'$ is polynomially simulated by $\mathcal{T}$ (for refutations of $\mathcal{T}'$, there are refutations of $\mathcal{T}$ of up to a polynomial same asymptotic size), but not vice versa. Two tableau calculi are *efficiency-incomparable* if neither one is polynomially simulated by the other.

3.4.1 Tableaux for Normal Logic Programs

In what follows, we provide infinite families of unsatisfiable normal programs witnessing that neither $\mathcal{T}_{nomore}$ is polynomially simulated by $\mathcal{T}_{smodels}$, nor vice versa. This means that, on certain normal programs, restricting *Cut* to only either atoms or bodies leads to exponentially greater (optimal) search space traversals of atom- or rule-based ASP solvers in

[17] In [65, 66], occurrences of aggregates are limited to rule bodies. Furthermore, the semantics proposed there is not based on the logic of here-and-there, since (default) negated aggregates like $not\ 0\{a\}0$ are treated differently. Hence, important properties that can be verified in the logic of here-and-there (e.g., strong equivalence [163]) do not carry forward to the semantics proposed in [65, 66].

[18] The determining factor for the asymptotic size of minimal refutations is the number of required *Cut* applications, that is, the number of branches that need to be investigated, because the depth of each branch is bounded by the input size. Also note that proof complexity says nothing about the difficulty of finding minimal refutations; rather, it provides a lower bound on the efficiency of proof-finding procedures, independent of heuristic influences.

$$\Pi_a^n = \left\{ \begin{array}{l} x \leftarrow not\ x \\ x \leftarrow a_1, b_1 \\ \quad \vdots \\ x \leftarrow a_n, b_n \end{array} \right\} \quad \Pi_b^n = \left\{ \begin{array}{ll} y \leftarrow c_1, \ldots, c_n, not\ y & \\ c_1 \leftarrow not\ a_1 & c_1 \leftarrow not\ b_1 \\ \quad \vdots & \quad \vdots \\ c_n \leftarrow not\ a_n & c_n \leftarrow not\ b_n \end{array} \right\} \quad \Pi_c^n = \left\{ \begin{array}{l} a_1 \leftarrow not\ b_1 \\ b_1 \leftarrow not\ a_1 \\ \quad \vdots \\ a_n \leftarrow not\ b_n \\ b_n \leftarrow not\ a_n \end{array} \right\}$$

Figure 3.9: Families $\{\Pi_a^n\}$, $\{\Pi_b^n\}$, and $\{\Pi_c^n\}$ of normal programs.

comparison to their counterparts, no matter the applied heuristics. The following results state the existence of witnessing families.

Proposition 3.15. *There is an infinite family $\{\Pi^n\}$ of normal programs such that*

1. *the size of minimal refutations of $\mathcal{T}_{nomore}$ for Π^n is asymptotically linear in n;*
2. *the size of minimal refutations of $\mathcal{T}_{smodels}$ for Π^n is asymptotically exponential in n.*

Proposition 3.16. *There is an infinite family $\{\Pi^n\}$ of normal programs such that*

1. *the size of minimal refutations of $\mathcal{T}_{smodels}$ for Π^n is asymptotically linear in n;*
2. *the size of minimal refutations of $\mathcal{T}_{nomore}$ for Π^n is asymptotically exponential in n.*

Family $\{\Pi_a^n \cup \Pi_c^n\}$ witnesses Proposition 3.15, and $\{\Pi_b^n \cup \Pi_c^n\}$ witnesses Proposition 3.16 (see Figure 3.9). The reason why $\mathcal{T}_{smodels}$ does not admit compact refutations for $\Pi_a^n \cup \Pi_c^n$ is that its proofs must exhaustively investigate symmetric alternatives obtained by cutting on atoms a_i or b_i. In fact, minimal refutations of $\mathcal{T}_{smodels}$ for $\Pi_a^n \cup \Pi_c^n$ are of the shape sketched in Figure 3.10. While an initial cut on x yields an immediate contradiction in the branch with $\boldsymbol{F}x$, branches with $\boldsymbol{T}x$ can only be completed after adding $n-1$ entries of the form $\boldsymbol{F}\{a_i, b_i\}$ for $1 \leq i \leq n$. However, *Cut*$[atom(\Pi_a^n \cup \Pi_c^n)]$ does not allow for introducing such entries, so that they must be deduced indirectly, extending branches obtained by cutting on atoms a_i or b_i. But cascaded applications of *Cut*$[atom(\Pi_a^n \cup \Pi_c^n)]$ yield a subtableau with 2^{n-1} branches below $\boldsymbol{T}x$ (and $\boldsymbol{F}\{not\ x\}$), whose leaves are indicated at the bottom of Figure 3.10. Given that exponentially many branches are required, the size of minimal refutations of $\mathcal{T}_{smodels}$ for $\Pi_a^n \cup \Pi_c^n$ is asymptotically exponential in n. Unlike this, the use of *Cut*$[body(\Pi_a^n \cup \Pi_c^n)]$ admits linear refutations for $\Pi_a^n \cup \Pi_c^n$ with $\mathcal{T}_{nomore}$, like the one sketched in Figure 3.11. In such a refutation, cuts on $\{not\ x\}$ or $\{a_i, b_i\}$ for $1 \leq i \leq n$ yield immediate contradictions in branches with $\boldsymbol{T}\{not\ x\}$ or $\boldsymbol{T}\{a_i, b_i\}$, respectively. In fact, only n applications of *Cut*$[body(\Pi_a^n \cup \Pi_c^n)]$ are required in total, so that there are linear refutations with $n+1$ branches overall.

The situation that $\mathcal{T}_{nomore}$ dominates $\mathcal{T}_{smodels}$ is reversed with programs of the family $\{\Pi_b^n \cup \Pi_c^n\}$, where Figure 3.12 sketches a minimal refutation of $\mathcal{T}_{nomore}$ for $\Pi_b^n \cup \Pi_c^n$. In this refutation, only the initial cut on $\{c_1, \ldots, c_n, not\ y\}$ yields an immediate contradiction in the branch with $\boldsymbol{T}\{c_1, \ldots, c_n, not\ y\}$. Then, cuts on rule bodies $\{not\ a_i\}$ (or, alternatively, $\{not\ b_i\}$) must be cascaded to deduce $\boldsymbol{T}c_i$ in each of the resulting branches. Only after $n-1$ entries of the form $\boldsymbol{T}c_i$ for $1 \leq i \leq n$ have been generated, the leaves indicated at the bottom of Figure 3.12 are obtained. In view of symmetry, the subtableau below $\boldsymbol{F}\{c_1, \ldots, c_n, not\ y\}$ (and $\boldsymbol{F}y$) necessarily includes 2^{n-1} branches, so that the size of minimal refutations of $\mathcal{T}_{nomore}$ for $\Pi_b^n \cup \Pi_c^n$ is asymptotically exponential in n. Unlike this, cuts on c_i, admitted with $\mathcal{T}_{smodels}$, yield immediate contradictions in branches with $\boldsymbol{F}c_i$, and the same applies to a branch with $\boldsymbol{T}y$. Thus, the refutation of $\mathcal{T}_{smodels}$ for

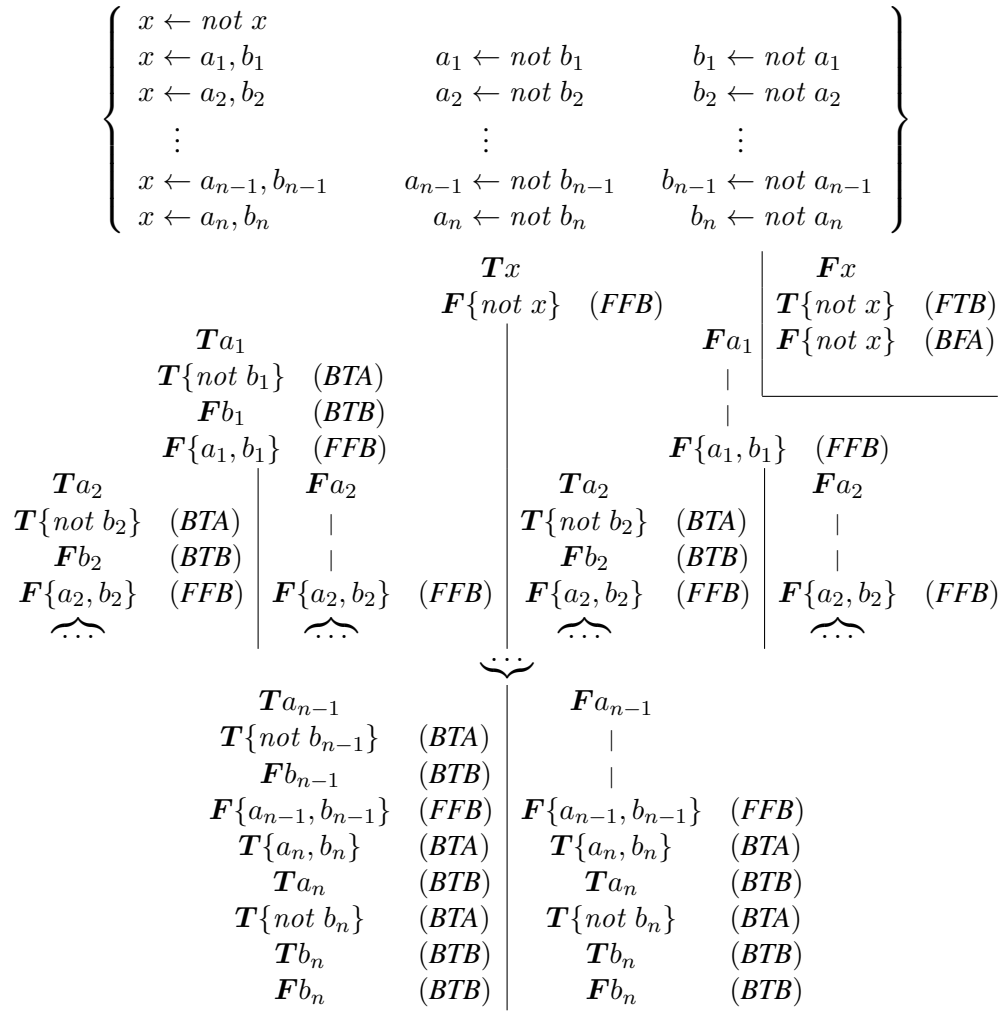


Figure 3.10: A minimal refutation of $\mathcal{T}_{smodels}$ for $\Pi_a^n \cup \Pi_c^n$, using $Cut[atom(\Pi_a^n \cup \Pi_c^n)]$.

$\Pi_b^n \cup \Pi_c^n$ sketched in Figure 3.13 involves only n applications of $Cut[atom(\Pi_b^n \cup \Pi_c^n)]$ in total, so that there are linear refutations with $n + 1$ branches overall.

Notably, empirical evidence for divergent proof complexities of $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ has been given in [2], and [110] shows that conflict resolution as performed by *smodels*-based ASP solver $smodels_{cc}$ is unable to compensate for the exponential proof complexity of $\mathcal{T}_{smodels}$ on family $\{\Pi_a^n \cup \Pi_c^n\}$. In view of mutual exponential separations, the next result is immediately obtained from Proposition 3.15 and 3.16.

Corollary 3.17. *Tableau calculi $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ are efficiency-incomparable.*

Given that refutations of $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ are refutations of $\mathcal{T}_{nomore++}$ as well, we have that $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ are both polynomially simulated by $\mathcal{T}_{nomore++}$. Hence, the following is an immediate consequence of Corollary 3.17.

Corollary 3.18. *Tableau calculus $\mathcal{T}_{nomore++}$ is exponentially stronger than both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$.*

The major implication of Corollary 3.18 is that, on certain normal programs, a priori restricting *Cut* to only either atoms or bodies necessitates exponentially greater search

$$\left\{\begin{array}{lll} x \leftarrow not\ x & & \\ x \leftarrow a_1, b_1 & a_1 \leftarrow not\ b_1 & b_1 \leftarrow not\ a_1 \\ x \leftarrow a_2, b_2 & a_2 \leftarrow not\ b_2 & b_2 \leftarrow not\ a_2 \\ \quad\vdots & \quad\vdots & \quad\vdots \\ x \leftarrow a_{n-1}, b_{n-1} & a_{n-1} \leftarrow not\ b_{n-1} & b_{n-1} \leftarrow not\ a_{n-1} \\ x \leftarrow a_n, b_n & a_n \leftarrow not\ b_n & b_n \leftarrow not\ a_n \end{array}\right\}$$

$\boldsymbol{T}\{not\ x\}$ | $\boldsymbol{F}\{not\ x\}$

$\boldsymbol{F}x$ (*BTB*) | $\boldsymbol{T}x$ (*BFB*)

$\boldsymbol{T}x$ (*FTA*) | $\boldsymbol{T}\{a_1, b_1\}$ | $\boldsymbol{F}\{a_1, b_1\}$

$\boldsymbol{T}a_1$ (*BTB*) | $\boldsymbol{T}\{a_2, b_2\}$ | $\boldsymbol{F}\{a_2, b_2\}$

$\boldsymbol{T}\{not\ b_1\}$ (*BTA*) | $\boldsymbol{T}a_2$ (*BTB*) | $\ddots$

$\boldsymbol{T}b_1$ (*BTB*) | $\boldsymbol{T}\{not\ b_2\}$ (*BTA*) | $\boldsymbol{F}\{a_{n-1}, b_{n-1}\}$

$\boldsymbol{F}b_1$ (*BTB*) | $\boldsymbol{T}b_2$ (*BTB*) | $\boldsymbol{T}\{a_n, b_n\}$ (*BTA*)

$\boldsymbol{F}b_2$ (*BTB*) | $\boldsymbol{T}a_n$ (*BTB*)

$\boldsymbol{T}\{not\ b_n\}$ (*BTA*)

$\boldsymbol{T}b_n$ (*BTB*)

$\boldsymbol{F}b_n$ (*BTB*)

Figure 3.11: A minimal refutation of $\mathcal{T}_{nomore}$ for $\Pi_a^n \cup \Pi_c^n$, using *Cut*$[body(\Pi_a^n \cup \Pi_c^n)]$.

space traversals than unrestricted *Cut*. Note that the phenomenon of exponentially greater proof complexity in comparison to $\mathcal{T}_{nomore++}$ does not, depending on the program family, apply to one of $\mathcal{T}_{smodels}$ or $\mathcal{T}_{nomore}$ alone. Rather, the infinite family

$$\left\{\begin{array}{l} (\Pi_a^n \setminus \{x \leftarrow not\ x\}) \cup (\Pi_b^n \setminus \{y \leftarrow c_1, \ldots, c_n, not\ y\}) \cup \\ \{y \leftarrow c_1, \ldots, c_n, not\ x, not\ y\} \cup \Pi_c^n \end{array}\right\}$$

is such that the asymptotic size of minimal refutations of both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ is exponential in n, while $\mathcal{T}_{nomore++}$ still admits refutations of linear size. Here, with $\mathcal{T}_{smodels}$, it is easy to prove that x needs to be true, while checking that this cannot be the case requires investigating symmetric alternatives by cutting on atoms a_i or b_i. On the other hand, with $\mathcal{T}_{nomore}$, it is easy to verify that x and y need to be false, but recognizing that $c_1, \ldots, c_n$ must be true, so that the rule $y \leftarrow c_1, \ldots, c_n, not\ x, not\ y$ is unsatisfied, requires exhaustive cutting on rule bodies $\{not\ a_i\}$ or $\{not\ b_i\}$. Unlike this, with $\mathcal{T}_{nomore++}$, it is easy to refute $c_1, \ldots, c_n$ to be false as well as x and y to be true, which in turn yields $y \leftarrow c_1, \ldots, c_n, not\ x, not\ y$ as unsatisfied rule. Hence, $\mathcal{T}_{nomore++}$, but neither $\mathcal{T}_{smodels}$ nor $\mathcal{T}_{nomore}$, admits linear refutations for members of the above family.

Note that our proof complexity results are unimpaired by failed-literal detection [76] as, e.g., applied by *smodels*. In fact, failed-literal detection can be mimicked by means of *Cut*, so that proof complexity, already assuming an optimal heuristic, stays unaffected. In view of Corollary 3.4 on Page 25 and similarities between *smodels* and *dlv* on normal programs [122], the proof complexity of tableau calculus $\mathcal{T}_{smodels}$ indeed provides a lower bound on the efficiency of *smodels* and *dlv* (when applied to normal programs).

3.4.2 Generic Tableaux for Composite Language Constructs

After considering normal programs and tableau calculi for them, we now turn to the generic calculus (cf. Figure 3.4 on Page 34) and extensions thereof. In view of Proposi-

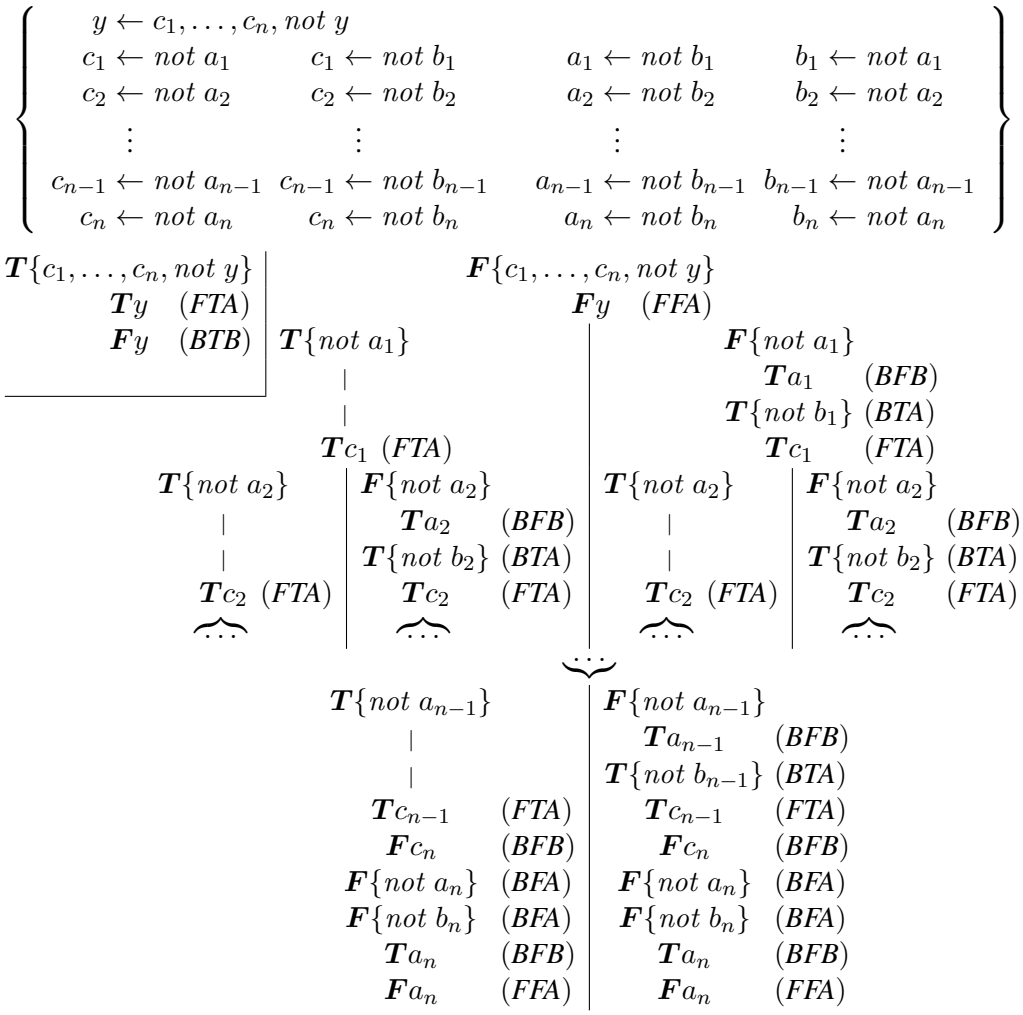


Figure 3.12: A minimal refutation of $\mathcal{T}_{nomore}$ for $\Pi_b^n \cup \Pi_c^n$, using $Cut[body(\Pi_b^n \cup \Pi_c^n)]$.

tion 3.12 on Page 38, the results in Section 3.4.1 (and the fact that minimal refutations of $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ for members of $\{\Pi_a^n \cup \Pi_c^n\}$ or $\{\Pi_b^n \cup \Pi_c^n\}$, respectively, are not significantly reduced when adding $WFJ[2^{atom(\Pi)}]$) allow us to immediately conclude that, for conjunctive programs, the generic calculi $\{$*(a)–(f)*, *(h)–(k)*, $Cut[atom(\Pi)]\}$ and $\{$*(a)–(f)*, *(h)–(k)*, $Cut[conj(\Pi)]\}$ are efficiency-incomparable, while $\{$*(a)–(f)*, *(h)–(k)*, $Cut[atom(\Pi) \cup conj(\Pi)]\}$ is exponentially stronger than both of them. We below extend the analysis of relative efficiency w.r.t. different *Cut* rules to more general logic programs, addressing the question whether cutting on further language constructs, namely, cardinality constraints and disjunctions, leads to more powerful tableau calculi. Beforehand, note that cutting on atoms is sufficient for obtaining complete calculi even in the presence of language extensions, given that the truth values of composite constructs can be deduced from atomic literals by (deterministic) tableau rules. For cardinality constraints and disjunctions, this is possible using the tableau rules in Figure 3.7 and 3.8 on Page 41 and 44, respectively.

For cardinality programs Π, we consider $\mathcal{T}_{card} = \{$*(a)–(f)*, *(h)–(r)*, $Cut[atom(\Pi) \cup conj(\Pi) \cup card(\Pi)]\}$ and $\mathcal{T}_{conj} = \{$*(a)–(f)*, *(h)–(r)*, $Cut[atom(\Pi) \cup conj(\Pi)]\}$. Both calculi contain all deterministic tableau rules dealing with cardinality programs; the difference is that cutting on cardinality constraints is allowed with $\mathcal{T}_{card}$, but not with $\mathcal{T}_{conj}$. As

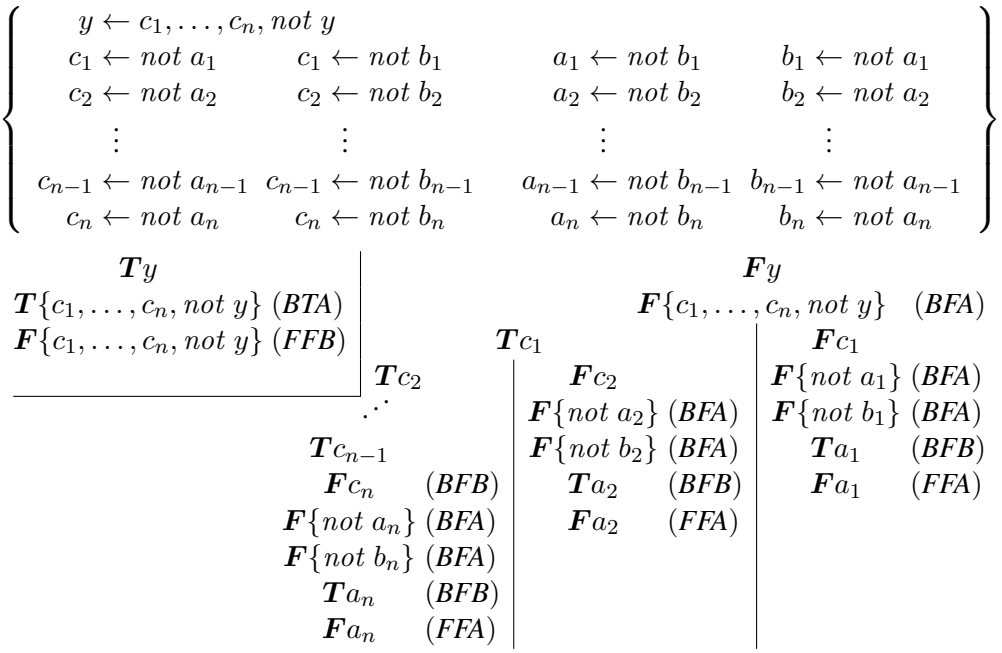


Figure 3.13: A minimal refutation of $\mathcal{T}_{smodels}$ for $\Pi_b^n \cup \Pi_c^n$, using *Cut*$[atom(\Pi_b^n \cup \Pi_c^n)]$.

every tableau of $\mathcal{T}_{conj}$ is a tableau of $\mathcal{T}_{card}$ as well, it is clear that $\mathcal{T}_{conj}$ is polynomially simulated by $\mathcal{T}_{card}$. The following result states that the converse does not hold.

Proposition 3.19. *Tableau calculus $\mathcal{T}_{card}$ is exponentially stronger than $\mathcal{T}_{conj}$.*

Proposition 3.19 is witnessed by the infinite family $\{\Pi_c^n \cup \Pi_d^n\}$ of unsatisfiable cardinality programs, where Π_c^n is shown in Figure 3.9 and Π_d^n is as follows:

$$\Pi_d^n \quad = \quad \{z \leftarrow 1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\}$$

For $\Pi_c^n \cup \Pi_d^n$, a branch containing $\boldsymbol{F}1\{a_i, b_i\}2$ is easy to refute because $\boldsymbol{F}a_i$ and $\boldsymbol{F}b_i$ can be deduced by tableau rule $TL_U{\downarrow}$ (cf. Figure 3.7 on Page 41), yielding an immediate contradiction since a_i and b_i cannot jointly be false (cf. Π_c^n in Figure 3.9). The unrestricted *Cut* rule of $\mathcal{T}_{card}$ can be used to exploit this by cutting on $1\{a_i, b_i\}2$ for $1 \leq i \leq n$, so that the resulting minimal refutations are of asymptotically linear size in n. In fact, when replacing cuts on y and c_i by cuts on z and $1\{a_i, b_i\}2$, respectively, minimal refutations of $\mathcal{T}_{card}$ for $\Pi_c^n \cup \Pi_d^n$ are of the shape sketched in Figure 3.13 (also assuming that applications of *BFA* to deduce $\boldsymbol{F}\{not\ a_i\}$ and $\boldsymbol{F}\{not\ b_i\}$ are replaced by $TL_U{\downarrow}$, deducing $\boldsymbol{F}a_i$ as well as $\boldsymbol{F}b_i$, and that $\boldsymbol{T}a_i$ is deduced by $I{\downarrow}$ instead of *BFB*). In contrast, with $\mathcal{T}_{conj}$, *Cut* must be applied to atoms a_i or b_i (or to bodies $\{not\ a_i\}$ or $\{not\ b_i\}$), while deducing $\boldsymbol{T}1\{a_i, b_i\}2$ in each of the resulting branches. Such refutations are of the same shape as the one sketched in Figure 3.12: an initial cut on $\{1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\}$ (or z) yields an immediate contradiction in the branch with $\boldsymbol{T}\{1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\}$ (or $\boldsymbol{T}z$) as well as a subtableau with 2^{n-1} branches below $\boldsymbol{F}\{1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\}$ (and $\boldsymbol{F}z$).

The practical consequence of Proposition 3.19 is that ASP solvers dealing with cardinality constraints can gain significant speed-ups by branching on them. Notably, the compilation of rules with cardinality constraints to so-called "basic constraint rules" [209], as

done by grounders like *lparse* [211] and *gringo* [90], introduces auxiliary atoms abbreviating cardinality constraints. This allows ASP solvers to (implicitly) branch on cardinality constraints, even if case analyses are restricted to atoms as, e.g., in *smodels*. Unlike this, our tableau framework does not rely on any compilation and considers cardinality constraints as structural entities that can be used deliberately for branching.

Regarding disjunctive programs, we have that occurrences of disjunctions are limited to heads of rules (to avoid involved definitions of $\overrightarrow{sup}$ and max). If this restriction were dropped, program Π_d^n could be rewritten using disjunctions $\{a_i; b_i\}$ rather than $1\{a_i, b_i\}2$ for $1 \leq i \leq n$. This would yield the same exponential separation w.r.t. *Cut* rules with and without disjunctions, respectively, as observed on cardinality constraints. However, with disjunctions $\{l_1; \ldots; l_n\}$ limited to heads of rules, the difficulty is that the information gained in the case of $\boldsymbol{T}\{l_1; \ldots; l_n\}$ is weak: it is exploited by tableau rule $FD{\downarrow}$ (cf. Figure 3.8 on Page 44) only if all but one of the literals $l_1, \ldots, l_n$ have already been assigned to false. In view of this, it is complicated (if at all possible) to come up with an infinite family of unsatisfiable disjunctive programs such that cutting on disjunctions is the source of an exponential separation. Hence, we leave the question open whether cutting on disjunctive heads admits exponentially smaller refutations than obtainable without it.

As noted in Section 3.2, existing ASP solvers, such as *smodels*, *nomore*, and *nomore++*, lack backward propagation for unfounded sets. Their associated tableau calculi reflect this by not including any variant of tableau rule *WFJ*$[\Omega]$ (cf. Figure 3.1 on Page 19) or $U{\downarrow}$ (cf. Figure 3.4 on Page 34), respectively. On the other hand, SAT-based and native conflict-driven learning ASP solvers, such as *assat*, *clasp*, *cmodels*, *sag*, and *smodels*$_{cc}$, are able to perform this kind of propagation relative to recorded loop formulas. This brings our attention to the question whether omitting some inferences deteriorates proof complexity. In what follows, we denote by $R{\uparrow}$ and $R{\downarrow}$ the forward and backward variant, respectively, of any of the (deterministic) tableau rules in Figure 3.4, 3.5, 3.7, and 3.8. For a tableau calculus $\mathcal{T}$, we say that $\mathcal{T}' \subseteq \mathcal{T}$ is an *approximation* of $\mathcal{T}$ if $\mathcal{T} \setminus \mathcal{T}' \subseteq \{R{\downarrow} \mid R{\uparrow} \in \mathcal{T}'\}$. (We assume that $TLU{\uparrow} \in \mathcal{T}'$ if $\{TL_U{\downarrow}, T_LU{\downarrow}\} \cap (\mathcal{T} \setminus \mathcal{T}') \neq \emptyset$, given that $TLU{\uparrow}$ has *two* backward counterparts.) That is, if $\mathcal{T}$ contains both $R{\uparrow}$ and $R{\downarrow}$, an approximation $\mathcal{T}'$ of $\mathcal{T}$ is allowed to drop $R{\downarrow}$. It is clear that every approximation $\mathcal{T}'$ of $\mathcal{T}$ is polynomially simulated by $\mathcal{T}$. Assuming *Cut* to be sufficiently powerful, the next result shows that the converse holds as well.

Proposition 3.20. *Let Π be a disjunctive program, $\mathcal{T}$ a tableau calculus containing any subset of the tableau rules (a)–(v), and $\mathcal{T}'$ an approximation of $\mathcal{T}$.*

If Cut$[\Gamma] \in \mathcal{T}'$ such that $atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \subseteq \Gamma$, then we have that $\mathcal{T}$ is polynomially simulated by $\mathcal{T}'$.

In fact, an inference due to $R{\downarrow}$ can be mimicked by cutting on the consequent of $R{\downarrow}$. Then, one of the two resulting branches becomes contradictory when applying $R{\uparrow}$. But recall that proof complexity assumes an optimal heuristic, determining the "right" objects to cut on. As an optimal heuristic is inaccessible in practice, it is certainly advantageous to implement $R{\downarrow}$ within an ASP solver whenever it can be done efficiently.

3.5 Related Work

Our work is inspired by the one of Järvisalo, Junttila, and Niemelä [144], who use tableau methods for investigating Boolean circuit satisfiability checking. Although their target is

different from ours, both approaches have aspects in common. First, both use tableau methods for characterizing DPLL-style search. Second, they analyze proof complexity w.r.t. cut rules characterizing different concepts of case analyses.

As pointed out in [131], DPLL is very similar to the propositional version of the KE tableau calculus; both are closely related to weak connection tableaux with atomic cut. Tableau-based characterizations of logic programming are elaborated upon in [73]. Pearce, de Guzmán, and Valverde [193] provide a tableau calculus for automated theorem proving in equilibrium logic. Further tableau approaches to non-monotonic reasoning are summarized in [191] and [49].

General investigations into propositional proof complexity [39], in particular, the one of (UN)SAT, can be found in [19]. Notably, recent results on CDCL [18, 197], the state-of-the-art complete algorithm for SAT solving, indicate its strong correlation to general resolution. Although DPLL amounts to a weaker form of resolution, called tree-like (cf. [19]), Järvisalo and Oikarinen [145] show that an extension of our basic tableau framework is as powerful as extended resolution (under standard translations between ASP and SAT, viz., completion and the reduction in [186]). The complexity considerations in [122] also build on the proximity of traditional ASP solving methods to DPLL.

Regarding inference systems for ASP, Bonatti [23] describes a resolution method for skeptical reasoning. Unlike our approach, it is based on query-oriented top-down evaluation, as also performed in SLDNF resolution (cf. [172]). Similarly, the proof schemes investigated by Marek and Remmel [173] are closely related to SLDNF resolution. Operator-based characterizations of propagation and choice techniques of ASP solvers can be found in [64, 209, 154, 3, 32]. They are more coarse-grained than our tableau rules, which aim at characterizing fundamental inference steps. Although SAT-based and native conflict-driven learning ASP solvers are usually described in terms of algorithms [167, 218, 123, 168, 95], the fact that they identify reasons for conflicts yields a close relationship to our tableau-based approach. In principle, (immediate) reasons can easily be extracted from tableau rules, albeit our tableau frameworks do not incorporate conflict-driven learning. The state-based calculus by Lierler [160], inspired by a similar approach [189] to Satisfiability Modulo Theories (SMT) [15], allows for characterizing several atom-based ASP solvers that incorporate conflict-driven learning.

A major issue in ASP solving is the treatment of unfounded sets [216, 159], which can be captured by loop formulas [167, 156]. As the number of (non-redundant) loop formulas may be exponential [164],[19] ASP solvers use dedicated procedures to check [209, 32] and possibly also extract [167, 123, 168, 4, 102, 52] (violated) loop formulas relative to assignments. To our knowledge, no existing ASP solver implements backward inference via tableau rule *WFJ*, unless loop formulas have been recorded. Unfortunately, the approach in this direction suggested in [33, 34] is computationally too complex (quadratic) to be beneficial in practice. Our generic tableau framework does not distinguish loops (in tableau rules $U\uparrow$ and $U\downarrow$), which could however be done based on loops for propositional theories [70]. Loops and loop formulas for first-order normal programs have been defined in [35, 157]. There are also direct characterizations (not referring to grounding) of answer sets or stable models, respectively, for first-order theories [194, 71]. To our knowledge, they have not yet been used as a basis for proof-theoretic frameworks.

[19] Lifschitz and Razborov [164] show that, under widely accepted assumptions in complexity theory, any semantics-preserving polynomial translation of normal programs to propositional theories must extend the input vocabulary. For instance, *lp2sat* [139] implements a sub-quadratic translation based on a binary representation of level mappings [136, 187] (which are considered w.r.t. "non-tight" programs).

3.6 Discussion

In contrast to the area of SAT, where the proof-theoretic foundations of SAT solvers are well-understood (cf. [19, 18, 197]), the literature on ASP solvers is generally too specific in terms of procedures or particular solving strategies. We addressed this deficiency by proposing tableau frameworks that provide us with formal means for characterizing and analyzing computations of ASP solvers. This is accomplished by associating specific tableau calculi with the approaches of ASP solvers, rather than their solving procedures. In fact, tableau calculi abstract from implementation details and admit identifying inference patterns. The latter can, in principle, be exploited to precisely render the constraints propagated by a solver in order to use them, e.g., for conflict-driven learning.

The explicit representation of rule bodies and further composite language constructs, such as cardinality constraints and disjunctions, in assignments has several benefits. For one, it allows us to characterize SAT-based and also native atom- or rule-based ASP solving approaches in a closer fashion. In fact, even in atom-based solvers, such as *dlv*, *smodels*, and $smodels_{cc}$, which (logically) work on assignments over atoms only, inferences rely on the valuations of rule bodies (cf. Section 3.2). Hence, the decision of whether or not to include composite language constructs in assignments mainly affects the available cut objects. In this respect, the consideration of atoms as well as rule bodies may lead to exponentially smaller (best-case) complexity than obtained with restricted approaches. This also applies to cardinality constraints in logic programs, while it is open whether branching on disjunctive heads can be the source of an exponential separation. The potential of exponential proof complexity decreases due to extending the range of cut objects is also confirmed by several related investigations (cf. [144, 145, 143]). However, it is well-known that uncontrolled cut applications are prone to inefficiency, and restricting them to (sub)formulas occurring in the input showed to be an effective way to "tame" the cut [40]. Our tableau calculi adopt such input restrictions, which is in line with the fact that current ASP solvers do not "invent" new cut objects.

The simple class of normal programs is, in principle, sufficient to represent all NP-problems in ASP [175], and it can be regarded as the core language shared by virtually all ASP solvers. On the other hand, practical experience shows that language extensions, such as *dlv*'s aggregates [67] or *smodels*' cardinality and weight constraints [209], are important for effective modeling (cf. [214]). Hence, we presented a generic tableau framework and illustrated its extension to composite language constructs on two examples: cardinality constraints and disjunctive heads. Independently of the construct under consideration, inference rules follow two major objectives: first, characterizing models of logic programs and, second, verifying that true atoms are non-circularly supported. Different notions of support are possible. For instance, atoms derived via composite language constructs in heads of rules may be subject to minimization, as with disjunctions, or not, as with cardinality constraints allowing for "choices." Such issues need to be settled in order to devise appropriate inference patterns.[20]

For conflict-driven learning ASP solvers, it is not only important to know valid inferences, but also how the propagated constraints look like. Our generic tableau framework provides means to study such aspects of composite language constructs in the course of specifying tableau rules for them, and it also provides a ready-to-use basis for check-

[20] The available language constructs also affect computational complexity; for instance, it increases by one level in the polynomial time hierarchy with disjunctive heads or negative weights within weight constraints (cf. [69]).

ing the soundness and completeness of sophisticated inference patterns. Interestingly, conditions allowing for the falsification of atoms that cannot be non-circularly supported inherently characterize unfounded sets for extended classes of logic programs. In particular, we are unaware of any pre-existing direct unfounded set definition (not relying on compilation to basic language constructs) for disjunctive programs admitting cardinality constraints in heads as well as bodies of rules, while the proviso of generic tableau rule $U\uparrow$ provides such a definition in view of Theorem 3.14. Based on our methodology, the consideration of unfounded sets could be extended to further composite language constructs.

Chapter 4

Conflict-Driven Answer Set Solving

Modern industrial SAT solvers utilize sophisticated lookback techniques (cf. [21]). Except for some early approaches [17, 179], lookback techniques obliging to the "First-UIP" scheme [219, 203, 48, 7, 197] have become consensus and are exploited by virtually all state-of-the-art SAT solvers based on CDCL [179, 185, 56].

In ASP, we distinguish between two kinds of solvers: native ones, such as *clasp* [95], *dlv* [158], *nomore++* [3], *smodels* [209], and *smodels*$_{cc}$ [218], and SAT-based solvers, such as *assat* [167], *cmodels* [123], and *sag* [168]. The latter apply SAT solvers to compute models of a logic program's completion [37, 6] and perform separate checks for unfounded sets [216, 159], whereas native ASP solvers internalize unfounded set checking as part of their propagation. Among them, traditional ASP solvers like *dlv*, *nomore++*, and *smodels* perform DPLL-style search, while the lookback techniques driving CDCL are supported by *clasp* and *smodels*$_{cc}$. To this end, *smodels*$_{cc}$ extends *smodels*' algorithm in specific ways, so that *clasp* remains as the first (and currently still the only) native ASP solver genuinely developed for conflict-driven ASP solving.

In this chapter, we present the logical fundament of *clasp* as well as its algorithmic foundations. The key idea is to view all inferences from logic programs as unit propagation on nogoods, stemming from completion and unfounded sets. This provides us with a uniform setting serving as the basis for conflict-driven ASP solving. However, nogoods stemming from unfounded sets are tested via a dedicated unfounded set checking routine and expatiated only "on demand." Beyond a CDCL-like procedure for deciding answer set existence, we devise novel algorithms for the enumeration of entire answer sets as well as their projections to a subset of "output" atoms. By virtue of dedicated backtracking, enumeration integrates with lookback techniques according to the First-UIP scheme, while still running in polynomial space.

The outline of this chapter is as follows. In Section 4.1, we characterize answer sets of normal programs as solutions for Boolean constraints. Section 4.2 introduces ordered assignments and unit propagation; these concepts are exploited by the algorithms provided in the sequel. In Section 4.3, we present our decision procedure for conflict-driven ASP solving along with its subroutines for propagation, unfounded set checking, and conflict analysis. Enumeration algorithms for answer sets as well as their projections are developed in Section 4.4. In Section 4.5, we present experimental results illustrating the effectiveness of our techniques in practice. Section 4.6 and 4.7 conclude the chapter by surveying related work and discussing the achieved results, respectively.

Parts of this chapter have also been presented in [94, 95, 98, 99], coauthored by the author of this thesis.

4.1 Nogoods of Normal Logic Programs

As already noted in Section 3.2.3, (deterministic) tableau rules like the ones in Figure 3.1 on Page 19 inherently induce nogoods (defined in Section 2.2), given that such a rule expresses the fact that its prerequisites necessarily imply its consequent. Investigating all instances of deterministic tableau rules w.r.t. a logic program Π thus allows for extracting a set Δ of nogoods such that any solution $\mathbf{A}$ for Δ belongs to a non-contradictory complete branch $(\Pi, \mathbf{A})$ in a tableau (and vice versa). Importantly, nogoods provide us with reasons explaining why entries must (not) belong to a solution, and lookback techniques can be used to analyze and recombine inherent reasons for conflicts.

The specification of Boolean constraints given below adheres to the distinction made in Section 3.2.3 between the completion, $Comp(\Pi)$, and the loop formulas, $LF(\Pi)$, of a normal program Π. As established in Theorem 3.5, models of $Comp(\Pi)$ match non-contradictory complete branches in tableaux of $\mathcal{T}_{comp}$, containing the deterministic tableau rules *(a)–(h)* in Figure 3.1. Furthermore, Theorem 3.1 and 3.8 have established that $\mathcal{T}_{comp}$ augmented with either *WFN*$[2^{atom(\Pi)}]$ or *WFN*$[loop(\Pi)]$ characterizes models of $Comp(\Pi) \cup LF(\Pi)$, as Lin and Zhao [167] showed that they coincide with the answer sets of Π. The major difference between $Comp(\Pi)$ and $LF(\Pi)$ is that the former captures local conditions that apply to individual atoms and rule bodies, while $LF(\Pi)$ aims at the more global conditions related to unfounded sets (cf. Section 2.3). Given that there may be exponentially many (non-redundant) loop formulas [164], ASP solvers do not a priori construct all of them explicitly, but check [209, 32] and possibly extract [167, 123, 168, 4, 102, 52] particular ones during answer set computation. The algorithms presented in later sections also pursue this approach.

In the following, we specify nogoods such that their solutions correspond to answer sets. To begin with, for a normal program Π, the set of *completion nogoods* of Π, denoted by Δ_Π, is as follows:

$$\begin{aligned}\Delta_\Pi \;=\; & \bigcup_{B \in body(\Pi), B=\{l_1,\ldots,l_n\}} \left\{ \begin{array}{l} \{\boldsymbol{F}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\}, \\ \{\boldsymbol{T}B, \boldsymbol{f}l_1\}, \ldots, \{\boldsymbol{T}B, \boldsymbol{f}l_n\} \end{array} \right\} \\ \cup \; & \bigcup_{p \in atom(\Pi), body_\Pi(p)=\{B_1,\ldots,B_k\}} \left\{ \begin{array}{l} \{\boldsymbol{T}p, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\}, \\ \{\boldsymbol{F}p, \boldsymbol{T}B_1\}, \ldots, \{\boldsymbol{F}p, \boldsymbol{T}B_k\} \end{array} \right\}\end{aligned}$$

As discussed in Section 3.2.3, Δ_Π can, for instance, be obtained by syntactically converting the tableau rules *(a)–(h)* in Figure 3.1 into nogoods. To be more precise, the nogood $\{\boldsymbol{F}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\}$ expresses the fact that a body B must not be assigned to false if all of its literals hold; the same exclusion is achieved by tableau rule *FTB* (or *BFB*, provided that $B \neq \emptyset$). The nogoods $\{\boldsymbol{T}B, \boldsymbol{f}l_1\}, \ldots, \{\boldsymbol{T}B, \boldsymbol{f}l_n\}$, representing that B cannot hold if some of its literals is false, comply with tableau rule *FFB* or, alternatively, *BTB*; if B is empty, i.e., if $n = 0$, there are no nogoods of this kind, and the corresponding tableau rules are likewise inapplicable. Turning to an atom p, the nogood $\{\boldsymbol{T}p, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\}$ stipulates p to be false if all of its supporting bodies are assigned to false; tableau rule *FFA* (or *BTA*, provided that $body_\Pi(p) \neq \emptyset$) expresses the same. Finally, the nogoods $\{\boldsymbol{F}p, \boldsymbol{T}B_1\}, \ldots, \{\boldsymbol{F}p, \boldsymbol{T}B_k\}$ view program rules as implications, complying with tableau rule *FTA* or, alternatively, *BFA*; if p has no supporting rule, i.e., if $k = 0$, there are no nogoods of this kind, and the corresponding tableau rules are likewise inapplicable.

Tableau Rules	Nogoods in Δ_{Π_6}
FTB, *BFB*	$\{\boldsymbol{F}\{not\ b\}, \boldsymbol{F}b\}, \{\boldsymbol{F}\{not\ a\}, \boldsymbol{F}a\}, \{\boldsymbol{F}\{a\}, \boldsymbol{T}a\}, \{\boldsymbol{F}\{d\}, \boldsymbol{T}d\}, \{\boldsymbol{F}\{c\}, \boldsymbol{T}c\}, \{\boldsymbol{F}\{c, not\ a\}, \boldsymbol{T}c, \boldsymbol{F}a\}$
FFB, *BTB*	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}, \{\boldsymbol{T}\{not\ a\}, \boldsymbol{T}a\}, \{\boldsymbol{T}\{a\}, \boldsymbol{F}a\}, \{\boldsymbol{T}\{d\}, \boldsymbol{F}d\}, \{\boldsymbol{T}\{c\}, \boldsymbol{F}c\}, \{\boldsymbol{T}\{c, not\ a\}, \boldsymbol{F}c\}, \{\boldsymbol{T}\{c, not\ a\}, \boldsymbol{T}a\}$
FTA, *BFA*	$\{\boldsymbol{F}a, \boldsymbol{T}\{not\ b\}\}, \{\boldsymbol{F}b, \boldsymbol{T}\{not\ a\}\}, \{\boldsymbol{F}c, \boldsymbol{T}\{a\}\}, \{\boldsymbol{F}c, \boldsymbol{T}\{d\}\}, \{\boldsymbol{F}d, \boldsymbol{T}\{c, not\ a\}\}, \{\boldsymbol{F}e, \boldsymbol{T}\{c\}\}, \{\boldsymbol{F}e, \boldsymbol{T}\{d\}\}$
FFA, *BTA*	$\{\boldsymbol{T}a, \boldsymbol{F}\{not\ b\}\}, \{\boldsymbol{T}b, \boldsymbol{F}\{not\ a\}\}, \{\boldsymbol{T}c, \boldsymbol{F}\{a\}, \boldsymbol{F}\{d\}\}, \{\boldsymbol{T}d, \boldsymbol{F}\{c, not\ a\}\}, \{\boldsymbol{T}e, \boldsymbol{F}\{c\}, \boldsymbol{F}\{d\}\}$

Table 4.1: Set Δ_{Π_6} of nogoods and associated tableau rules of $\mathcal{T}_{comp}$ for Π_6.

Example 4.1. *Reconsider Π_6 from Example 3.7:*

$$\Pi_6 \quad = \quad \left\{ \begin{array}{ll} r_1: & a \leftarrow not\ b \\ r_2: & b \leftarrow not\ a \\ r_3: & c \leftarrow a \\ r_4: & c \leftarrow d \\ r_5: & d \leftarrow c, not\ a \\ r_6: & e \leftarrow c \\ r_7: & e \leftarrow d \end{array} \right\}$$

We have that $atom(\Pi_6) = \{a, b, c, d, e\}$, $body(\Pi_6) = \{\{not\ b\}, \{not\ a\}, \{a\}, \{d\}, \{c\}, \{c, not\ a\}\}$, and the tableau rules (a)–(h) in Figure 3.1 correspond to nogoods in Δ_{Π_6} as shown in Table 4.1. Since each body occurring in Π_6 is non-empty and each atom has a supporting rule, every nogood in Δ_{Π_6} reflects exactly one forward- and one backward-oriented tableau rule. As already illustrated in Example 3.8 on Page 28, the nogoods in Δ_{Π_6} can be derived syntactically by considering potential applications of the tableau rules (a)–(h) in Figure 3.1 for each target $v \in atom(\Pi_6) \cup body(\Pi_6)$.

In view of the syntactic relationship between the tableau rules in Figure 3.1 and Δ_Π, we derive the following counterpart of Theorem 3.5 on Page 26 in terms of nogoods.[1]

Proposition 4.1. *Let Π be a normal program and $X \subseteq atom(\Pi) \cup body(\Pi)$.*

Then, we have that $(X \cap atom(\Pi)) \cup \{p_B \mid B \in X \cap body(\Pi)\}$ is a model of $Comp(\Pi)$ iff $\{\boldsymbol{T}v \mid v \in X\} \cup \{\boldsymbol{F}v \mid v \in (atom(\Pi) \cup body(\Pi)) \setminus X\}$ is a solution for Δ_Π.

Example 4.2. *Reconsider Π_6 and its associated nogoods Δ_{Π_6}, shown in Example 4.1 and Table 4.1, respectively. The completion $Comp(\Pi_6)$ is as follows:*

$$Comp(\Pi_6) \quad = \quad \left\{ \begin{array}{ll} a \leftrightarrow p_{\{not\ b\}} & p_{\{not\ b\}} \leftrightarrow \neg b \\ b \leftrightarrow p_{\{not\ a\}} & p_{\{not\ a\}} \leftrightarrow \neg a \\ c \leftrightarrow p_{\{a\}} \vee p_{\{d\}} & p_{\{a\}} \leftrightarrow a \\ & p_{\{d\}} \leftrightarrow d \\ d \leftrightarrow p_{\{c, not\ a\}} & p_{\{c, not\ a\}} \leftrightarrow c \wedge \neg a \\ e \leftrightarrow p_{\{c\}} \vee p_{\{d\}} & p_{\{c\}} \leftrightarrow c \end{array} \right\}$$

The models of $Comp(\Pi_6)$ and corresponding solutions for Δ_{Π_6} are shown in Table 4.2.

[1] In the following, we sometimes write solutions as sets of entries, rather than sequences.

Model of $Comp(\Pi_6)$	Solution for Δ_{Π_6}
$\{a, c, e\} \cup$ $\{p_{\{not\ b\}}, p_{\{a\}}, p_{\{c\}}\}$	$\{\boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{T}c, \boldsymbol{F}d, \boldsymbol{T}e\} \cup$ $\{\boldsymbol{T}\{not\ b\}, \boldsymbol{F}\{not\ a\}, \boldsymbol{T}\{a\}, \boldsymbol{F}\{d\},$ $\boldsymbol{T}\{c\}, \boldsymbol{F}\{c, not\ a\}\}$
$\{b\} \cup$ $\{p_{\{not\ a\}}\}$	$\{\boldsymbol{F}a, \boldsymbol{T}b, \boldsymbol{F}c, \boldsymbol{F}d, \boldsymbol{F}e\} \cup$ $\{\boldsymbol{F}\{not\ b\}, \boldsymbol{T}\{not\ a\}, \boldsymbol{F}\{a\}, \boldsymbol{F}\{d\},$ $\boldsymbol{F}\{c\}, \boldsymbol{F}\{c, not\ a\}\}$
$\{b, c, d, e\} \cup$ $\{p_{\{not\ a\}}, p_{\{d\}}, p_{\{c\}}, p_{\{c,not\ a\}}\}$	$\{\boldsymbol{F}a, \boldsymbol{T}b, \boldsymbol{T}c, \boldsymbol{T}d, \boldsymbol{T}e\} \cup$ $\{\boldsymbol{F}\{not\ b\}, \boldsymbol{T}\{not\ a\}, \boldsymbol{F}\{a\}, \boldsymbol{T}\{d\},$ $\boldsymbol{T}\{c\}, \boldsymbol{T}\{c, not\ a\}\}$

Table 4.2: Models of $Comp(\Pi_6)$ and corresponding solutions for Δ_{Π_6}.

Since the completion of a program Π defines propositions standing for bodies in terms of atoms (and atoms in terms of propositions for bodies), we have that bodies' truth values are unambiguously determined if all atoms are assigned. In fact, solutions for the subset of Δ_Π defining bodies can be described as follows.

Lemma 4.2. *Let Π be a normal program and $X \subseteq atom(\Pi)$.*
Then, we have that

$$\begin{aligned}\mathbf{A} \;=\; & \{\boldsymbol{T}p \mid p \in X\} \cup \{\boldsymbol{F}p \mid p \in atom(\Pi) \setminus X\} \\ \cup\; & \{\boldsymbol{T}B \mid B \in body(\Pi), B^+ \subseteq X, B^- \cap X = \emptyset\} \\ \cup\; & \{\boldsymbol{F}B \mid B \in body(\Pi), (B^+ \cap (atom(\Pi) \setminus X)) \cup (B^- \cap X) \neq \emptyset\}\end{aligned}$$

is the unique solution for

$$\bigcup_{B \in body(\Pi), B=\{l_1,\ldots,l_n\}} \{\{\boldsymbol{F}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\}, \{\boldsymbol{T}B, \boldsymbol{f}l_1\}, \ldots, \{\boldsymbol{T}B, \boldsymbol{f}l_n\}\} \;\subseteq\; \Delta_\Pi$$

such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$.

An important subclass of logic programs, called "tight" [8, 60], consists of programs without positive recursion among atoms. We say that a normal program Π is *tight* if $loop(\Pi) = \emptyset$. For such programs, it is well-known [68] that answer sets coincide with models of the completion. By combining Theorem 3.5 on Page 26 and Theorem 3.8 on Page 27 with Proposition 4.1 and Lemma 4.2, we can reformulate this result in terms of solutions for Δ_Π.

Theorem 4.3. *Let Π be a tight program and $X \subseteq atom(\Pi)$.*
Then, we have that X is an answer set of Π iff

$$\begin{aligned}\mathbf{A} \;=\; & \{\boldsymbol{T}p \mid p \in X\} \cup \{\boldsymbol{F}p \mid p \in atom(\Pi) \setminus X\} \\ \cup\; & \{\boldsymbol{T}B \mid B \in body(\Pi), B^+ \subseteq X, B^- \cap X = \emptyset\} \\ \cup\; & \{\boldsymbol{F}B \mid B \in body(\Pi), (B^+ \cap (atom(\Pi) \setminus X)) \cup (B^- \cap X) \neq \emptyset\}\end{aligned}$$

is the unique solution for Δ_Π such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$.

Example 4.3. *Program Π_6, shown in Example 4.1, is non-tight, given that $loop(\Pi_6) = \{\{c, d\}\}$ in view of r_4 and r_5. However, $\Pi_6 \setminus \{r_4\}$ is tight, and one can check that the first two (but not the third) models of $Comp(\Pi_6)$, as shown in Table 4.2, are models of $Comp(\Pi_6 \setminus \{r_4\})$ as well. By Theorem 4.3, the corresponding solutions for $\Delta_{\Pi_6 \setminus \{r_4\}}$ (the first two solutions shown in Table 4.2) comprise the answer sets $\{a, c, e\}$ and $\{b\}$ of $\Pi_6 \setminus \{r_4\}$.*

In order to characterize also the answer sets of non-tight programs Π by solutions for nogoods, we need to reflect tableau rule *WFN*$[loop(\Pi)]$ or *WFN*$[2^{atom(\Pi)}]$ (cf. Figure 3.1). To this end, we define the set of *loop nogoods* of Π, denoted by Λ_Π, as follows:

$$\Lambda_\Pi \;=\; \bigcup_{U \subseteq atom(\Pi), EB_\Pi(U)=\{B_1,\ldots,B_k\}} \{\{\boldsymbol{T}p, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \mid p \in U\}$$

The nogoods in Λ_Π express that an atom p must not be assigned to true if it belongs to an unfounded set U; the same exclusion is achieved by tableau rule *WFN*$[2^{atom(\Pi)}]$ (or *WFJ*$[2^{atom(\Pi)}]$, provided that $EB_\Pi(U) \neq \emptyset$).

In view of the correspondence between (deterministic) tableau rules in, e.g., $\mathcal{T}_{nomore++}$ (defined in Section 3.1) and $\Delta_\Pi \cup \Lambda_\Pi$, along with Lemma 4.2, we derive the following counterpart of Theorem 3.1 on Page 20 in terms of nogoods.

Theorem 4.4. *Let Π be a normal program and $X \subseteq atom(\Pi)$.*

Then, we have that X is an answer set of Π iff

$$\begin{aligned}\mathbf{A} \;=\;& \{\boldsymbol{T}p \mid p \in X\} \cup \{\boldsymbol{F}p \mid p \in atom(\Pi) \setminus X\}\\ \cup\;& \{\boldsymbol{T}B \mid B \in body(\Pi), B^+ \subseteq X, B^- \cap X = \emptyset\}\\ \cup\;& \{\boldsymbol{F}B \mid B \in body(\Pi), (B^+ \cap (atom(\Pi) \setminus X)) \cup (B^- \cap X) \neq \emptyset\}\end{aligned}$$

is the unique solution for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$.

Example 4.4. *Reconsider Π_6, shown in Example 4.1, and the solutions for Δ_{Π_6}, as shown in Table 4.2. One can check that the first two of these solutions satisfy also the loop nogoods in Λ_{Π_6}. Hence, by Theorem 4.4, the sets of their true atoms, $\{a, c, e\}$ and $\{b\}$, respectively, are answer sets of Π_6. Unlike this, the loop nogoods $\{\boldsymbol{T}c, \boldsymbol{F}\{a\}\}$ and $\{\boldsymbol{T}d, \boldsymbol{F}\{a\}\}$, obtained for $U = \{c, d\}$ in view of $EB_{\Pi_6}(\{c, d\}) = \{\{a\}\}$, are (amongst others) violated by the third solution for Δ_{Π_6}. This tells us that the set of atoms true in the third solution for Δ_{Π_6}, $\{b, c, d, e\}$, is not an answer set of Π_6.*

By Theorem 4.4, the nogoods in $\Delta_\Pi \cup \Lambda_\Pi$ describe a set of constraints that need to be checked for identifying answer sets. However, while the size of Δ_Π is linear in the size of Π, the one of Λ_Π is, in general, exponential. As shown by Lifschitz and Razborov [164], the latter is not a defect in the construction of Λ_Π, but an implication of widely accepted assumptions in complexity theory. Hence, most answer set solvers work on logic programs as succinct representations of loop nogoods (or formulas, respectively) and check them efficiently by determining unfounded sets relative to assignments. To this end, program structure, namely, (elementary) loops [167, 156, 109, 103], can be used to confine unfounded set checking to necessary parts. In fact, the algorithms presented below exploit such program structure, making use of the results in Section 2.3.

4.2 Ordered Assignments and Unit Propagation

While the order of entries in a Boolean assignment does not affect its semantics, it is crucial for algorithms analyzing the entries' interdependencies. Hence, we in the following assume that, for an assignment $\mathbf{A} = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i, \ldots, \sigma_n)$, each entry $\sigma_i \in \mathbf{A}$ for $1 \leq i \leq n$ has an associated *decision level*, a non-negative integer denoted by $dlevel(\sigma_i)$. Furthermore, we let $\mathbf{A}[\sigma_i] = (\sigma_1, \ldots, \sigma_{i-1})$ denote the *prefix* of $\mathbf{A}$ relative to σ_i, while defining $\mathbf{A}[\sigma] = \mathbf{A}$ if $\sigma \notin \mathbf{A}$. Given this, we call $\mathbf{A} = (\sigma_1, \ldots, \sigma_n)$ an *ordered* (*Boolean*) *assignment* if, for every $1 \leq i \leq n$, it holds that

1. $var(\sigma_i) \notin \{var(\sigma) \mid \sigma \in \mathbf{A}[\sigma_i]\}$ and
2. $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}[\sigma_i]\} \cup \{0\}) \leq dlevel(\sigma_i)$.

The first condition stipulates distinct entries in $\mathbf{A}$ to assign distinct variables, which implies that ordered assignments are non-contradictory. The second condition requires decision levels to be monotonically increasing along the sequence of entries in $\mathbf{A}$.

For an ordered assignment $\mathbf{A} = (\sigma_1, \ldots, \sigma_{i-1}, \sigma_i, \ldots, \sigma_n)$ and an entry σ with an associated decision level, $dlevel(\sigma)$, we denote the *insertion* of σ into $\mathbf{A}$ by $\mathbf{A} \circ \sigma = (\sigma_1, \ldots, \sigma_{i-1}, \sigma, \sigma_i, \ldots, \sigma_n)$, where

1. $\max(\{dlevel(\sigma_1), \ldots, dlevel(\sigma_{i-1})\} \cup \{0\}) \leq dlevel(\sigma)$ and
2. $\min(\{dlevel(\sigma_i), \ldots, dlevel(\sigma_n)\} \cup \{dlevel(\sigma) + 1\}) = dlevel(\sigma) + 1$.

That is, $\mathbf{A} \circ \sigma$ contains σ as the last entry with decision level smaller than $dlevel(\sigma) + 1$. For instance, inserting $\boldsymbol{F}d$ into $\mathbf{A} = (\boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{T}c, \boldsymbol{T}e, \boldsymbol{F}f)$ yields $\mathbf{A} \circ \boldsymbol{F}d = (\boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{T}c, \boldsymbol{F}d, \boldsymbol{T}e, \boldsymbol{F}f)$ when $dlevel(\boldsymbol{T}a) = 0$, $dlevel(\boldsymbol{F}b) = dlevel(\boldsymbol{T}c) = dlevel(\boldsymbol{F}d) = 1$, and $dlevel(\boldsymbol{T}e) = dlevel(\boldsymbol{F}f) = 2$. In Section 4.3 and 4.4.1, insertions always append entries to the end of assignments, while they can also be inserted into assignments' middle parts with the enumeration algorithms presented in Section 4.4.2 and 4.4.3.

Given a nogood δ and an assignment $\mathbf{A}$, an entry σ is *unit-resulting* for δ w.r.t. $\mathbf{A}$ if

1. $\delta \setminus \mathbf{A} = \{\overline{\sigma}\}$ and
2. $\sigma \notin \mathbf{A}$.

The first condition stipulates $\overline{\sigma}$ to be the only entry of δ not contained in $\mathbf{A}$, which implies that a violated nogood does not have any unit-resulting entry. The second condition precludes duplicates: if $\mathbf{A}$ already contains σ, then it cannot be unit-resulting. For instance, $\boldsymbol{F}d$ is unit-resulting for the nogood $\{\boldsymbol{F}b, \boldsymbol{T}d\}$ w.r.t. the assignment $(\boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{T}c)$, but neither w.r.t. $(\boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{T}c, \boldsymbol{T}d)$ nor $(\boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{T}c, \boldsymbol{F}d)$. Note that the notion of a unit-resulting entry is closely related to unit clauses considered in SAT solving (cf. [21]). Along the lines of SAT, we call the iterated process of extending an assignment by unit-resulting entries *unit propagation*. An algorithm performing unit propagation on nogoods induced by a normal program is presented in Section 4.3.2.

For an ordered assignment $\mathbf{A}$ and an entry σ, we call a nogood δ an *antecedent* of σ w.r.t. $\mathbf{A}$ if σ is unit-resulting for δ w.r.t. $\mathbf{A}[\sigma]$. Furthermore, for a set Δ of nogoods, we say that σ is *implied* by Δ w.r.t. $\mathbf{A}$ if Δ contains some antecedent of σ w.r.t. $\mathbf{A}$. We extend this notion to a decision level dl in the following way: dl is *implied* by Δ w.r.t. $\mathbf{A}$ if every entry $\sigma \in \mathbf{A}$ such that $\max(\{dlevel(\rho) \mid \rho \in \mathbf{A}[\sigma]\} \cup \{0\}) = dlevel(\sigma) = dl$ is implied by Δ w.r.t. $\mathbf{A}$. The concept of being implied identifies entries that must necessarily be included in $\mathbf{A}$ because the complement of any of them would immediately yield some violated nogood (an antecedent). When considering decision levels, all but the first entry at a level dl need to be implied for dl being implied; if $dl = 0$, any first entry $\sigma \in \mathbf{A}$ such that $dlevel(\sigma) = 0$ must be implied as well. Implied entries and decision levels are crucial for the meaningful application of conflict analysis, described in Section 4.3.4.

Note that assignments constructed by means of the decision procedure described next are such that all decision levels are implied, while the enumeration algorithms in Section 4.4.2 and 4.4.3 admit exceptions to this. When given a normal program Π, throughout this chapter, we assume that all variables occurring in an associated set of nogoods belong to $atom(\Pi) \cup body(\Pi)$; in particular, this applies to dynamic nogoods, which are below denoted by ∇.

4.3 Decision Algorithm

Given the specification of answer sets in terms of nogoods provided in Section 4.1, we can make use of sophisticated lookback techniques from SAT solving (cf. [21]) for developing equally advanced ASP solving procedures. But while SAT deals with plain nogoods, represented by clauses, our algorithms work on logic programs, inducing several kinds of nogoods. In fact, the nogoods in Section 4.1 provide semantic conditions (clauses are syntactic representations), and we do not assume any particular syntactic representation here. However, note that the exponentially many loop nogoods stemming from unfounded sets are succinctly given by a logic program, and the algorithms devised below determine individual ones only when used for unfounded set falsification. The main purpose of associating nogoods with a logic program is to provide reasons for entries derived by (unit) propagation. This puts ASP solving on the same logical fundament as SAT solving, so that similar reasoning strategies can be applied, while neither relying on a translation to SAT nor any proprietary techniques (apart from unfounded set checking).

In Section 4.3.1, we introduce our main conflict-driven ASP solving procedure. Section 4.3.2 details its subroutine for propagation. Our algorithm for unfounded set detection, which is the main particularity of ASP (compared to SAT), is presented in Section 4.3.3. Section 4.3.4 describes resolution-based conflict analysis in our setting. Finally, in Section 4.3.5, we outline the derivation of soundness and completeness results.

4.3.1 Conflict-Driven Nogood Learning

Our main procedure for deciding whether a normal program has an answer set is similar to CDCL with First-UIP scheme (cf. [179, 219, 56, 21]). In fact, clauses can be viewed as particular syntactic representations of nogoods, but other representations (e.g., gates, inequalities, rules, etc.) can be used as well. Hence, to abstract from syntax, we present our conflict-driven learning algorithm for deciding answer set existence in terms of nogoods and, in the following, call it *Conflict-Driven Nogood Learning* for ASP (CDNL-ASP).

Given a normal program Π, CDNL-ASP, shown in Algorithm 4.1, starts from an empty assignment $\mathbf{A}$ and an empty set ∇ of recorded nogoods over $atom(\Pi) \cup body(\Pi)$. The latter set is used to accumulate conflict and loop nogoods, which along with the completion nogoods in Δ_Π are exploited for (unit) propagation and conflict analysis. Moreover, by means of the decision level dl, initialized with 0, we count the number of *decision entries* in $\mathbf{A}$. Such entries are heuristically selected (in Line 14), while entries derived by propagation (in Line 5) are implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$.

For computing an answer set of Π or reporting that there is none, the main loop in Line 4–16 of Algorithm 4.1 follows the standard proceeding of CDCL. First, NOGOOD-PROPAGATION (detailed in Section 4.3.2) deterministically extends $\mathbf{A}$ in Line 5, and possibly also records loop nogoods from Λ_Π in ∇. Afterwards, one of the following cases applies:

Conflict. If propagation led to the violation of some nogood $\varepsilon \in \Delta_\Pi \cup \nabla$, as checked in Line 6, there are two possibilities. Either the conflict occurred independently of any previous decision, meaning that the input program Π has no answer set, or CONFLICT-ANALYSIS (detailed in Section 4.3.4) is performed in Line 8 to determine a conflict nogood δ, recorded in ∇ in Line 9, along with a decision level dl to jump back to. Note that we assume δ to be *asserting*, i.e., some entry must be unit-resulting for δ after

Algorithm 4.1: CDNL-ASP

Input : A normal program Π.
Output : An answer set of Π or "no answer set."

```
1  A := ∅                                   // ordered assignment over atom(Π) ∪ body(Π)
2  ∇ := ∅                                   // set of recorded nogoods
3  dl := 0                                  // decision level
4  loop
5      (A, ∇) := NogoodPropagation(Π, ∇, A)
6      if ε ⊆ A for some ε ∈ Δ_Π ∪ ∇ then   // conflict
7          if max({dlevel(σ) | σ ∈ ε} ∪ {0}) = 0 then return no answer set
8          (δ, dl) := ConflictAnalysis(ε, Π, ∇, A)
9          ∇ := ∇ ∪ {δ}                     // (temporarily) record conflict nogood
10         A := A \ {σ ∈ A | dl < dlevel(σ)}   // backjumping
11     else if A^T ∪ A^F = atom(Π) ∪ body(Π) then   // answer set
12         return A^T ∩ atom(Π)
13     else
14         σ_d := Select(Π, ∇, A)           // decision
15         dlevel(σ_d) := dl := dl + 1
16         A := A ∘ σ_d
```

backjumping in Line 10. This condition, which is guaranteed by ConflictAnalysis, makes sure that, after backjumping, CDNL-ASP traverses the search space differently from before (without explicitly flipping any decision entry).

Solution. If propagation led to a total assignment $\mathbf{A}$ (not violating any nogood in $\Delta_\Pi \cup \nabla$), as checked in Line 11, the atoms that are true in $\mathbf{A}$ belong to an answer set of Π, which is returned in Line 12.

Decision. If neither of the previous cases applies, $\mathbf{A}$ is partial, and a decision entry σ_d is selected according to some heuristic in Line 14. We do not make any particular assumptions about the heuristic used, but require that $var(\sigma_d) \in (atom(\Pi) \cup body(\Pi)) \setminus (\mathbf{A}^T \cup \mathbf{A}^F)$. That is, the variable in σ_d must be unassigned and occur in the input program Π. Also note that $dlevel(\sigma_d)$ is set to the increment of dl in Line 15,[2] so that σ_d is appended to the end of $\mathbf{A}$ in Line 16.

Example 4.5. *Reconsider Π_2 from Example 2.2:*

$$\Pi_2 = \left\{ \begin{array}{ll} r_1: & a \leftarrow not\ b \\ r_2: & b \leftarrow not\ a \\ r_3: & c \leftarrow a \\ r_4: & c \leftarrow b, d \\ r_5: & d \leftarrow b, c \\ r_6: & d \leftarrow e \\ r_7: & e \leftarrow b, not\ a \\ r_8: & e \leftarrow c, d \end{array} \right\}$$

[2]Notations of the form "$x_1 := \cdots := x_n := v$" mean that a value v is assigned to each of $x_1, \ldots, x_n$.

Although we have not yet detailed the subroutines used in Algorithm 4.1, let us consider a full-fledged computation of the answer set $\{b, c, d, e\}$ *of* Π_2*. To this end, Table 4.3 shows the current assignment* $\mathbf{A}$ *at different stages of* CDNL-ASP(Π_2)*, where columns provide the value of* dl*, viz., the current decision level, and the line of Algorithm 4.1 at which particular contents of* $\mathbf{A}$ *and/or some nogood* δ *are inspected. The entries inserted into* $\mathbf{A}$ *in Line 16 of Algorithm 4.1 are decision entries. Unlike them, each entry inserted into* $\mathbf{A}$ *in Line 5, that is, in an execution of* NOGOODPROPAGATION*, is unit-resulting for some nogood* $\delta \in \Delta_{\Pi_2} \cup \nabla$*; the info column displays which group of nogoods includes* δ*. Furthermore, we indicate successes of the test for a violated nogood performed in Line 6, and we show the nogood* δ *to be recorded in* ∇ *along with the decision level* dl *to jump back to (as info) as they are returned by* CONFLICTANALYSIS *when invoked in Line 8.*

In detail, a computation of CDNL-ASP(Π_2) *can start by successively picking decision entries* $\boldsymbol{T}d$*,* $\boldsymbol{F}\{b, not\ a\}$*,* $\boldsymbol{T}c$*, and* $\boldsymbol{F}\{not\ a\}$ *at levels* 1*,* 2*,* 3*, and* 4*, respectively. Observe that there is exactly one decision entry per level, and each decision is immediately followed by a propagation step, performed before making the next decision. At the start, propagation cannot derive any entry at decision levels* 1 *and* 2*, and thus assignment* $\mathbf{A}$ *stays partial. After the third decision, the entries shown below the horizontal (single) line are unit-resulting for respective nogoods* $\delta \in \Delta_{\Pi_2}$ *w.r.t.* $\mathbf{A}$*. Hence, they are inserted into* $\mathbf{A}$ *at decision level* 3*. Since* $\mathbf{A}$ *is still partial, decision entry* $\boldsymbol{F}\{not\ a\}$ *is picked at level* 4*. The following propagation step yields a total assignment, which is a solution for* Δ_{Π_2}*. However, we have that* $\{d, e\}$ *is unfounded for* Π_2 *w.r.t.* $\mathbf{A}$*, that is, the corresponding loop nogoods* $\{\boldsymbol{T}d, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$ *and* $\{\boldsymbol{T}e, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$ *are violated. Such violations are detected within* NOGOODPROPAGATION *and lead to the recording of some loop nogood from* Λ_{Π_2} *in* ∇*. In Table 4.3, we assume that* $\{\boldsymbol{T}d, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$ *is recorded, so that a conflict is encountered in Line 6 of Algorithm 4.1. Note that* $\boldsymbol{F}\{b, c\}$ *is the single entry of the nogood assigned at decision level* 4*. Hence,* $\{\boldsymbol{T}d, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$ *is instantly asserting and returned by* CONFLICTANALYSIS *in Line 8; the smallest decision level such that, after backjumping,* $\boldsymbol{T}\{b, c\}$ *is unit-resulting for* $\{\boldsymbol{T}d, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$ *is* 2*. The peculiarity that* CONFLICTANALYSIS *may be launched with an asserting (loop) nogood results from the "unidirectional" propagation of loop nogoods in current ASP solvers: as discussed in Section 3.2.3, ASP solvers implement tableau rule WFN, but not its contrapositive WFJ, although both tableau rules are logically based on loop nogoods. (We further comment on this phenomenon in Section 4.3.4.)*

Upon backjumping to decision level 2*, all entries inserted into* $\mathbf{A}$ *at levels* 3 *and* 4 *are retracted, and only the (decision) entries* $\boldsymbol{T}d$ *and* $\boldsymbol{F}\{b, not\ a\}$ *assigned at levels* 1 *and* 2 *are retained. In the sequel, the asserting nogood* $\{\boldsymbol{T}d, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$ *in* ∇ *enables the derivation of further entries by unit propagation, which results in another conflict, this time on the completion nogood* $\{\boldsymbol{T}\{not\ a\}, \boldsymbol{T}a\}$*. Starting from it,* CONFLICTANALYSIS *determines the asserting nogood* $\{\boldsymbol{F}\{b, not\ a\}, \boldsymbol{T}d\}$*. As a consequence,* CDNL-ASP(Π_2) *returns to decision level* 1*, where* $\boldsymbol{T}\{b, not\ a\}$ *is unit-resulting for* $\{\boldsymbol{F}\{b, not\ a\}, \boldsymbol{T}d\}$*. A final propagation step leads to a total assignment not violating any nogood in* $\Delta_{\Pi_2} \cup \nabla$*. (Notably, the nogoods in* Λ_{Π_2} *are left implicit and merely tested within* NOGOODPROPAGATION *via an unfounded set checking subroutine.) Entries in the obtained solution that comprise true atoms are underlined in Table 4.3. The associated answer set of* Π_2*,* $\{b, c, d, e\}$*, is returned as the result of* CDNL-ASP(Π_2)*.*

dl	$\mathbf{A}$	δ	Info	Line
1	$\boldsymbol{T}d$			16
2	$\boldsymbol{F}\{b, not\ a\}$			16
3	$\boldsymbol{T}c$			16
	$\boldsymbol{T}\{c, d\}$	$\{\boldsymbol{F}\{c, d\}, \boldsymbol{T}c, \boldsymbol{T}d\}$	Δ_{Π_2}	5
	$\boldsymbol{T}e$	$\{\boldsymbol{F}e, \boldsymbol{T}\{c, d\}\}$	Δ_{Π_2}	5
	$\boldsymbol{T}\{e\}$	$\{\boldsymbol{F}\{e\}, \boldsymbol{T}e\}$	Δ_{Π_2}	5
4	$\boldsymbol{F}\{not\ a\}$			16
	$\boldsymbol{T}a$	$\{\boldsymbol{F}\{not\ a\}, \boldsymbol{F}a\}$	Δ_{Π_2}	5
	$\boldsymbol{T}\{a\}$	$\{\boldsymbol{F}\{a\}, \boldsymbol{T}a\}$	Δ_{Π_2}	5
	$\boldsymbol{T}\{not\ b\}$	$\{\boldsymbol{T}a, \boldsymbol{F}\{not\ b\}\}$	Δ_{Π_2}	5
	$\boldsymbol{F}b$	$\{\boldsymbol{T}b, \boldsymbol{F}\{not\ a\}\}$	Δ_{Π_2}	5
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{T}\{b, c\}, \boldsymbol{F}b\}$	Δ_{Π_2}	5
	$\boldsymbol{F}\{b, d\}$	$\{\boldsymbol{T}\{b, d\}, \boldsymbol{F}b\}$	Δ_{Π_2}	5
		$\{\boldsymbol{T}d, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$	Λ_{Π_2}	6
		$\{\boldsymbol{T}d, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$	dl=2	8
2	$\boldsymbol{F}\{b, not\ a\}$			
	$\boldsymbol{T}\{b, c\}$	$\{\boldsymbol{T}d, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$	∇	5
	$\boldsymbol{T}b$	$\{\boldsymbol{T}\{b, c\}, \boldsymbol{F}b\}$	Δ_{Π_2}	5
	$\boldsymbol{T}a$	$\{\boldsymbol{F}\{b, not\ a\}, \boldsymbol{T}b, \boldsymbol{F}a\}$	Δ_{Π_2}	5
	$\boldsymbol{T}\{not\ a\}$	$\{\boldsymbol{T}b, \boldsymbol{F}\{not\ a\}\}$	Δ_{Π_2}	5
		$\{\boldsymbol{T}\{not\ a\}, \boldsymbol{T}a\}$	Δ_{Π_2}	6
		$\{\boldsymbol{F}\{b, not\ a\}, \boldsymbol{T}d\}$	dl=1	8
1	$\underline{\boldsymbol{T}d}$			
	$\boldsymbol{T}\{b, not\ a\}$	$\{\boldsymbol{F}\{b, not\ a\}, \boldsymbol{T}d\}$	∇	5
	$\underline{\boldsymbol{T}b}$	$\{\boldsymbol{T}\{b, not\ a\}, \boldsymbol{F}b\}$	Δ_{Π_2}	5
	$\boldsymbol{F}a$	$\{\boldsymbol{T}\{b, not\ a\}, \boldsymbol{T}a\}$	Δ_{Π_2}	5
	$\boldsymbol{T}\{not\ a\}$	$\{\boldsymbol{T}b, \boldsymbol{F}\{not\ a\}\}$	Δ_{Π_2}	5
	$\boldsymbol{F}\{not\ b\}$	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}$	Δ_{Π_2}	5
	$\boldsymbol{F}\{a\}$	$\{\boldsymbol{T}\{a\}, \boldsymbol{F}a\}$	Δ_{Π_2}	5
	$\underline{\boldsymbol{T}e}$	$\{\boldsymbol{F}e, \boldsymbol{T}\{b, not\ a\}\}$	Δ_{Π_2}	5
	$\boldsymbol{T}\{e\}$	$\{\boldsymbol{F}\{e\}, \boldsymbol{T}e\}$	Δ_{Π_2}	5
	$\boldsymbol{T}\{b, d\}$	$\{\boldsymbol{F}\{b, d\}, \boldsymbol{T}b, \boldsymbol{T}d\}$	Δ_{Π_2}	5
	$\underline{\boldsymbol{T}c}$	$\{\boldsymbol{F}c, \boldsymbol{T}\{b, d\}\}$	Δ_{Π_2}	5
	$\boldsymbol{T}\{b, c\}$	$\{\boldsymbol{F}\{b, c\}, \boldsymbol{T}b, \boldsymbol{T}c\}$	Δ_{Π_2}	5
	$\boldsymbol{T}\{c, d\}$	$\{\boldsymbol{F}\{c, d\}, \boldsymbol{T}c, \boldsymbol{T}d\}$	Δ_{Π_2}	5

Table 4.3: A computation of answer set $\{b, c, d, e\}$ with CDNL-ASP(Π_2).

Algorithm 4.2: NOGOODPROPAGATION

Input : A normal program Π, a set ∇ of nogoods, and an ordered assignment $\mathbf{A}$.
Output : An extended ordered assignment and set of nogoods.

```
1  U := ∅                                                          // unfounded set
2  loop
3      repeat
4          if δ ⊆ A for some δ ∈ Δ_Π ∪ ∇ then return (A, ∇)        // conflict
5          Σ := {δ ∈ Δ_Π ∪ ∇ | δ \ A = {σ̄}, σ ∉ A}                 // unit-resulting nogoods
6          if Σ ≠ ∅ then let σ̄ ∈ δ \ A for some δ ∈ Σ in
7              dlevel(σ) := max({dlevel(ρ) | ρ ∈ δ \ {σ̄}} ∪ {0})
8              A := A ∘ σ
9      until Σ = ∅
10     if loop(Π) = ∅ then return (A, ∇)      // no unfounded set ∅ ⊂ U ⊆ atom(Π) \ A^F
11     U := U \ A^F
12     if U = ∅ then U := UNFOUNDEDSET(Π, A)
13     if U = ∅ then return (A, ∇)            // no unfounded set ∅ ⊂ U ⊆ atom(Π) \ A^F
14     let p ∈ U in
15         ∇ := ∇ ∪ {{Tp} ∪ {FB | B ∈ EB_Π(U)}}  // (temporarily) record loop nogood
```

4.3.2 Nogood Propagation

The subroutine for deterministically extending an (ordered) assignment $\mathbf{A}$, NOGOODPROPAGATION, is shown in Algorithm 4.2. It combines unit propagation on completion nogoods in Δ_Π and recorded nogoods in ∇ (Line 3–9) with unfounded set checking (Line 10–15). While unit propagation is always run to a fixpoint (or a conflict), sophisticated unfounded set checks are performed only if the input program Π is non-tight. Otherwise, if Π is tight, Theorem 2.9 on Page 15 applies and, in view of $loop(\Pi) = \emptyset$, tells us that all unfounded sets U are already falsified, i.e., $U \subseteq \mathbf{A}^{\boldsymbol{F}}$ holds. In fact, when finishing the loop in Line 3–9, an assignment $\mathbf{A}$ at hand is both atom- and body-saturated for Π (cf. Definition 2.3 and 2.4 on Page 13 and 15, respectively), so that the results in Section 2.3 serve as a basis for demand-driven unfounded set checking via the subroutine UNFOUNDEDSET (detailed in Section 4.3.3). Also note that, by the construction of Σ in Line 5, we have that $var(\sigma)$ is unassigned before inserting an entry σ into $\mathbf{A}$ in Line 8. Along with the fact that $dlevel(\sigma)$ is set appropriately in Line 7, this makes sure that the (extended) assignment $\mathbf{A}$ computed by NOGOODPROPAGATION is ordered and that all newly assigned entries σ are implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$.

Example 4.6. *The main idea of integrating unfounded set checking with unit propagation is to trigger the successive falsification of unfounded atoms by recording loop nogoods from Λ_Π in ∇. To see this, consider a normal program Π containing the following rules:*

$$x \leftarrow y, z$$
$$y \leftarrow x$$
$$z \leftarrow y$$

Let $\mathbf{A}$ be an atom-saturated assignment such that $U = \{x, y, z\}$ is unfounded for Π w.r.t. $\mathbf{A}$ and $U \cap (\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}}) = \emptyset$. Then, we have that $EB_\Pi(U) \subseteq \mathbf{A}^{\boldsymbol{F}}$, so that $\boldsymbol{F}x$, $\boldsymbol{F}y$, and $\boldsymbol{F}z$ are unit-resulting for loop nogoods $\{\boldsymbol{T}p, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\}$ in Λ_Π, where $p \in U$

and $EB_\Pi(U) = \{B_1, \ldots, B_k\}$. *While neither* $\boldsymbol{F}x$, $\boldsymbol{F}y$, *nor* $\boldsymbol{F}z$ *may be unit-resulting for any completion nogood in* Δ_Π, *all of them (along with* $\boldsymbol{F}\{x\}$, $\boldsymbol{F}\{y\}$, *and* $\boldsymbol{F}\{y, z\}$*) are derived by unit propagation when given* $\Delta_\Pi \cup \{\{\boldsymbol{T}x, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\}\}$. *That is, when adding only one loop nogood from* Λ_Π *to* ∇, *the whole unfounded set* U *is falsified by unit propagation. However, whether the addition of a single loop nogood is sufficient to falsify a whole unfounded set depends on the structure of* Π. *For instance, when we augment* Π *with* $y \leftarrow z$, *the derivation of* $\boldsymbol{F}y$ *and* $\boldsymbol{F}z$ *by unit propagation is no longer certain because the (circular) supports* $y \leftarrow z$ *and* $z \leftarrow y$ *may not be eliminated by assigning* x *to false. We still derive* $\boldsymbol{F}\{x\}$, *i.e., the rule* $y \leftarrow x$ *becomes inapplicable, so that* $EB_\Pi(\{y, z\}) \subseteq (\mathbf{A} \cup \{\boldsymbol{F}x, \boldsymbol{F}\{x\}\})^{\boldsymbol{F}}$. *This shows that* $U \setminus (\mathbf{A} \cup \{\boldsymbol{F}x, \boldsymbol{F}\{x\}\})^{\boldsymbol{F}} = \{x, y, z\} \setminus \{x\} = \{y, z\}$ *remains as a smaller unfounded set.*

The observations made in Example 4.6 motivate the strategy of Algorithm 4.2 to successively falsify the elements of an unfounded set U. At the start, no (non-empty) unfounded set has been determined, and so U is initialized to be empty in Line 1. Provided that the loop in Line 3–9 finishes without conflict and that Π is non-tight, we remove all false atoms from U in Line 11. In the first iteration of the outer loop in Line 2–15, U stays empty, and our subroutine for unfounded set detection (detailed in Section 4.3.3) is queried in Line 12. The crucial assumption made here is that UNFOUNDEDSET($\Pi, \mathbf{A}$) returns an unfounded set $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ such that U is non-empty if some non-empty subset of $atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ is unfounded. Then, if a non-empty U is returned, the addition of a loop nogood $\{\boldsymbol{T}p\} \cup \{\boldsymbol{F}B \mid B \in EB_\Pi(U)\}$ to ∇ for an arbitrary $p \in U$, done in Line 15, yields either a conflict or the unit-resulting entry $\boldsymbol{F}p$ in the next iteration of the loop in Line 2–15. In the latter case, further elements of U may be falsified by unit propagation within the loop in Line 3–9. When we afterwards reconsider the previously determined unfounded set U, the removal of false atoms in Line 11 is guaranteed to result in another (smaller) unfounded set $U \setminus \mathbf{A}^{\boldsymbol{F}}$. Hence, if $U \setminus \mathbf{A}^{\boldsymbol{F}}$ is non-empty (checked in Line 12 before computing any new unfounded set), NOGOODPROPAGATION proceeds by adding the next loop nogood to ∇, which as before yields either a conflict or a unit-resulting entry. In this way, once a non-empty unfounded set U has been detected, it is falsified element by element; only after expending all elements of U, a new unfounded set is to be computed. All in all, NOGOODPROPAGATION terminates as soon as a conflict is encountered (in Line 4) or with an assignment such that no non-empty subset of $atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ is unfounded. If Π is tight, the latter is immediately verified in Line 10. Otherwise, the subroutine UNFOUNDEDSET, queried in Line 12, failed to detect a non-empty unfounded set (of non-false atoms) before finishing in Line 13.

Example 4.7. *To illustrate how* NOGOODPROPAGATION *utilizes nogoods, reconsider the computation of* CDNL-ASP(Π_2) *shown in Table 4.3. All implied entries, that is, the ones assigned below any (single) line at a decision level* dl, *are unit-resulting for nogoods in* $\Delta_{\Pi_2} \cup \nabla$ *and successively derived by unit propagation. In particular, at decision level* 4, *the implied entries* σ *have antecedents* $\delta \in \Delta_{\Pi_2}$ *such that all entries of* δ *except for* $\overline{\sigma}$ *are already contained in* $\mathbf{A}$ *when* σ *is inserted into* $\mathbf{A}$. *The impact of loop nogoods in* Λ_{Π_2} *can be observed on the conflict encountered at decision level* 4. *Here, we have that* $U = \{d, e\} \subseteq \mathbf{A}^{\boldsymbol{T}}$ *is unfounded, so that* $\mathbf{A}$ *violates each of the loop nogoods* $\{\boldsymbol{T}d, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$ *and* $\{\boldsymbol{T}e, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$. *After detecting the unfounded set* U *and recording* $\{\boldsymbol{T}d, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$ *in* ∇, *its violation gives rise to leaving* NOGOODPROPAGATION *in Line 4 of Algorithm 4.2.*

In summary, our subroutine for propagation interleaves unit propagation with the recording of loop nogoods. The latter is done only if the input program is non-tight and if the falsity of unfounded atoms cannot be derived by unit propagation via other available nogoods. Clearly, our approach favors unit propagation over unfounded set computations, which can be motivated as follows. For one, unit propagation does not contribute new dynamic nogoods to ∇, so that it is more "economic" than unfounded set checking. For another, although unfounded set detection algorithms (like the one provided below) are of linear time complexity, they need to analyze a logic program in a global fashion and may get stuck half-way, inspecting significant program parts without eventually detecting any non-empty unfounded set (of non-false atoms). But given that unfounded set checking (w.r.t. total assignments) is mandatory for soundness and (w.r.t. partial assignments) also helps to detect inherent conflicts early, the respective subroutine described next is nonetheless an integral part of NOGOODPROPAGATION.

4.3.3 Unfounded Set Checking

The subroutine for unfounded set detection, UNFOUNDEDSET, is invoked on a non-tight program Π when unit propagation reaches a fixpoint without any conflict nor extant unfalsified unfounded atoms (cf. Algorithm 4.2). As a matter of fact, a fixpoint of unit propagation is both atom- and body-saturated for Π. Hence, Corollary 2.10 on Page 15 applies and allows us to focus on unfounded sets of non-false atoms contained in non-trivial strongly connected components of $DG(\Pi)$ (the dependency graph of Π, defined in Section 2.3). To this end, for any $p \in atom(\Pi)$, let $scc(p)$ denote the set of all atoms belonging to the same strongly connected component as p in $DG(\Pi)$. We say that p is *cyclic* if its strongly connected component of $DG(\Pi)$ is non-trivial (that is, if there is some rule $r \in \Pi$ such that $head(r) \in scc(p)$ and $body(r)^+ \cap scc(p) \neq \emptyset$), and *acyclic* otherwise. In view of the results in Section 2.3, unfounded set checking can concentrate exclusively on cyclic atoms, since only they can belong to (unfounded) loops.[3]

Beyond static information about strongly connected components, UNFOUNDEDSET makes use of *source pointers* [209] to indicate non-circular supports of atoms. Given a normal program Π, the idea is to associate every (cyclic) $p \in atom(\Pi)$ with an element of $body_\Pi(p)$ (or one of the special-purpose symbols $\bot$ and $\top$), denoted by $source(p)$, pointing to a chain of rules witnessing that p cannot be unfounded. Hence, as long as $source(p)$ remains "intact," p can be excluded from unfounded set checks. In this way, source pointers enable lazy, incremental unfounded set checking relative to recent changes of an assignment. To make sure that still no unfounded set is missed, the following invariants need to be guaranteed:

1. For every cyclic $p \in atom(\Pi)$, we require that $source(p) \in body_\Pi(p) \cup \{\bot\}$.

2. The subgraph of $DG(\Pi)$ containing every cyclic $p \in atom(\Pi)$ along with edges (p, q) for all $q \in source(p)^+ \cap scc(p)$ must be acyclic.[4]

For a normal program Π, we call the collection of links $source(p)$ for all $p \in atom(\Pi)$ a *source pointer configuration*. We say that a source pointer configuration

[3] Strongly connected components of dependency graphs are also exploited by other unfounded set detection procedures [209, 32, 4] of native ASP solvers. We further discuss relationships to them in Section 4.6.

[4] For the special-purpose symbols $\bot$ and $\top$, we let $\bot^+ = \top^+ = \emptyset$.

Algorithm 4.3: UNFOUNDEDSET

Input : A normal program Π and an ordered assignment $\mathbf{A}$.
Output : An unfounded set of Π w.r.t. $\mathbf{A}$.

```
1  S := {p ∈ atom(Π) \ A^F | source(p) ∈ A^F ∪ {⊥}}          // initialize scope S
2  repeat
3  |  T := {p ∈ atom(Π) \ (A^F ∪ S) | source(p)^+ ∩ (scc(p) ∩ S) ≠ ∅}
4  |  S := S ∪ T                                              // extend scope S
5  until T = ∅
6  while S ≠ ∅ do let p ∈ S in                                // select starting point p
7  |  U := {p}
8  |  repeat
9  |  |  if EB_Π(U) ⊆ A^F then return U      // unfounded set ∅ ⊂ U ⊆ atom(Π) \ A^F
10 |  |  let B ∈ EB_Π(U) \ A^F in
11 |  |  |  if B^+ ∩ (scc(p) ∩ S) = ∅ then                   // shrink U
12 |  |  |  |  foreach q ∈ U such that B ∈ body_Π(q) do
13 |  |  |  |  |  source(q) := B
14 |  |  |  |  |  U := U \ {q}
15 |  |  |  |  |  S := S \ {q}
16 |  |  |  else U := U ∪ (B^+ ∩ (scc(p) ∩ S))                // extend U
17 |  until U = ∅
18 return ∅                                  // no unfounded set ∅ ⊂ U ⊆ atom(Π) \ A^F
```

is *valid* if it satisfies the aforementioned invariants. For an appropriate initialization, we define the *initial source pointer configuration* for Π by:

$$source(p) \quad = \quad \begin{cases} \bot & \text{if } p \in atom(\Pi) \text{ is cyclic} \\ \top & \text{if } p \in atom(\Pi) \text{ is acyclic} \end{cases}$$

While $\top$ expresses that an acyclic atom p does not need to be linked to any element of $body_\Pi(p)$, $\bot$ indicates that a non-circular support for a cyclic atom p still needs to be determined. We assume that the initial source pointer configuration for Π, which is valid by definition, is in place upon an invocation of CDNL-ASP(Π) (or one of the enumeration algorithms in Section 4.4). Moreover, in Section 4.3.5, we argue that UNFOUNDEDSET maintains the validity of source pointer configurations, which is crucial for completeness.

Given a normal program Π and an (ordered) assignment $\mathbf{A}$, UNFOUNDEDSET, shown in Algorithm 4.3, starts by collecting non-false (cyclic) atoms p whose source pointers are false ($source(p) \in \mathbf{A}^{\boldsymbol{F}}$) or yet undetermined ($source(p) = \bot$) in Line 1, as the possibility of non-circularly supporting such atoms is in question. In Line 2–5, this set is successively extended by adding atoms whose source pointers (positively) rely on it, thus providing the *scope* S for the second part of unfounded set checking. In fact, the loop in Line 6–17 aims at reestablishing source pointers for the atoms in S via rules whose bodies do not (positively) rely on S, so that these rules can provide non-circular support. Conversely, if source pointers cannot be reestablished, an unfounded set is detected.

In more detail, as long as the scope S is non-empty, an arbitrary atom $p \in S$ is picked in Line 6 of Algorithm 4.3 as starting point for the construction of a non-empty unfounded set U. If $EB_\Pi(U) \subseteq \mathbf{A}^{\boldsymbol{F}}$ holds in Line 9, the unfounded set U is immediately returned, so that NOGOODPROPAGATION can successively falsify its atoms by unit propagation (cf. Algorithm 4.2). Otherwise, some external body $B \in EB_\Pi(U) \setminus \mathbf{A}^{\boldsymbol{F}}$ is selected in

Line 10 for further investigation. If B^+ contains atoms in the scope S that belong to the same strongly connected component of $DG(\Pi)$ as the starting point p (checked in Line 11), we add them to U in Line 16, which makes B non-external w.r.t. the extended set U. On the other hand, if such atoms do not exist in B^+, it means that B can non-circularly support all of its associated head atoms $q \in U$. Then, in Line 12–15, the source pointers of such atoms q are set to B, and the atoms q are removed from both the unfounded set U under construction and the scope S. The described process continues until either U becomes empty (checked in Line 17), in which case the remaining atoms of S are investigated, or a (non-empty) unfounded set U is detected and returned in Line 9. Finally, if the scope S runs empty, source pointers could be reestablished for all atoms that had been contained in S, and UNFOUNDEDSET returns the empty unfounded set in Line 18.

In order to provide further intuitions, let us stress some major design principles of our unfounded set detection algorithm:

1. At each stage of the loop in Line 6–17, all atoms of U belong to $scc(p)$, where p is an atom added first to U (in Line 7). This is because further atoms, added to U in Line 16, are elements of $scc(p)$. (However, $U \subseteq scc(p)$ does not necessarily imply $p \in U$ for a (non-empty) unfounded set U returned in Line 9.)

2. At each stage of the loop in Line 6–17, we have that $U \subseteq S$, as all atoms added to U in either Line 7 or 16 belong to S. Hence, it holds that $q \in S$ whenever $source(q)$ is set to an (external) body $B \in body_\Pi(q)$ in Line 13, while $B^+ \cap (scc(p) \cap S) = \emptyset$ has been checked before (in Line 11). This makes sure that setting $source(q)$ to B does not introduce any cycles via source pointers.

3. Once detected, a (non-empty) unfounded set U is immediately returned in Line 9, and NOGOODPROPAGATION takes care of falsifying all elements of U before checking for any further unfounded set (cf. Algorithm 4.2). This reduces overlaps with unit propagation on the completion nogoods in Δ_Π, as it already handles unsupported atoms, i.e., singleton unfounded sets (and bodies relying on them).

4. The source pointer of an atom q in some unfounded set U returned in Line 9 needs not and is not reset to $\bot$. (In fact, $source(q)$ is only set in Line 13 when reestablishing a potential non-circular support for q.) Rather, we admit $source(q) \in \mathbf{A}^{\boldsymbol{F}}$ as long as $q \in \mathbf{A}^{\boldsymbol{F}}$, derived within NOGOODPROPAGATION upon falsifying U. Thus, when q becomes unassigned later on (after backjumping), $source(q)$ still allows for lazy unfounded set checking.

Example 4.8. *Let us illustrate Algorithm 4.3 on some invocations of* UNFOUNDEDSET($\Pi_2, \mathbf{A}$) *made upon the computation of the answer set* $\{b, c, d, e\}$ *of* Π_2 *described in Example 4.5. To this end, in Table 4.4, we indicate internal states of* UNFOUNDEDSET($\Pi_2, \mathbf{A}$) *when queried w.r.t. fixpoints* $\mathbf{A}$ *of unit propagation at decision levels* 0, 2, *and* 4, *respectively. Beforehand, note that* $scc(c) = scc(d) = scc(e) = \{c, d, e\}$, *while* a *and* b *are acyclic. Hence, before the first invocation of* UNFOUNDEDSET($\Pi_2, \mathbf{A}$) *at decision level* 0, *we have that* $source(a) = source(b) = \top$ *and* $source(c) = source(d) = source(e) = \bot$. *In view of Line 1 of Algorithm 4.3, we thus obtain the scope* $S = \{c, d, e\}$. *Then, assume that atom* e *is picked in Line 6 and added to* U *in Line 7, and that* $\{c, d\} \in EB_{\Pi_2}(\{e\})$ *is selected in Line 10. Since* $\{c, d\} \cap (scc(e) \cap S) = \{c, d\}$, *this*

dl	$source(p)$	S	U	$B \in EB_{\Pi_2}(U) \setminus \mathbf{A}^{\boldsymbol{F}}$	Line
0		$\{c,d,e\}$			1
		$\{c,d,e\}$	$\{e\}$		7
		$\{c,d,e\}$	$\{c,d,e\}$	$\{c,d\}$	16
	$source(e)$	$\{c,d\}$	$\{c,d\}$	$\{b, not\ a\}$	13
	$source(d)$	$\{c\}$	$\{c\}$	$\{e\}$	13
	$source(c)$	$\emptyset$	$\boxed{\emptyset}$	$\{b,d\}$	13
2	$\boldsymbol{F}\{b, not\ a\}$	$\{e\}$			1
	$\{e\}$	$\{d,e\}$			4
	$\{b,d\}$	$\{c,d,e\}$			4
		$\{c,d,e\}$	$\{d\}$		7
		$\{c,d,e\}$	$\{c,d\}$	$\{b,c\}$	16
	$source(c)$	$\{d,e\}$	$\{d\}$	$\{a\}$	13
	$source(d)$	$\{e\}$	$\emptyset$	$\{b,c\}$	13
		$\{e\}$	$\{e\}$		7
	$source(e)$	$\emptyset$	$\boxed{\emptyset}$	$\{c,d\}$	13
4	$\boldsymbol{F}\{b,c\}$	$\{d\}$			1
	$\{c,d\}$	$\{d,e\}$			4
		$\{d,e\}$	$\{e\}$		7
		$\{d,e\}$	$\boxed{\{d,e\}}$	$\{c,d\}$	16

Table 4.4: Runs of UNFOUNDEDSET($\Pi_2, \mathbf{A}$) upon a computation of answer set $\{b,c,d,e\}$.

makes us augment U with both c and d in Line 16, resulting in an intermediate state such that $U = \{c,d,e\}$. Further assume that $\{b, not\ a\} \in EB_{\Pi_2}(\{c,d,e\})$ is selected next in Line 10, for which $\{b\} \cap (scc(e) \cap S) = \emptyset$ holds in Line 11. Hence, $source(e)$ is set to $\{b, not\ a\}$ in Line 13, and e is removed from U and S in Line 14 and 15, respectively. In the same manner, $source(d)$ and $source(c)$ can in the following iterations of the loop in Line 8–17 be set to $\{e\}$ and $\{b,d\}$, respectively. Afterwards, we have that $U = S = \emptyset$, so that the empty unfounded set, surrounded by a box in Table 4.4, is returned (in Line 18). Given that there is no non-empty unfounded set, no entry is derived by unit propagation at decision level 0, as also indicated by omitting this level in Table 4.3.

The invocation of UNFOUNDEDSET($\Pi_2, (\boldsymbol{T}d)$) *at decision level 1 is not shown in Table 4.4, as it yields an empty scope S. Unlike this, with* UNFOUNDEDSET(Π_2, $(\boldsymbol{T}d, \boldsymbol{F}\{b, not\ a\})$) *at decision level 2, we have that $source(e) = \{b, not\ a\} \in \mathbf{A}^{\boldsymbol{F}}$, so that $S = \{e\}$ is obtained in Line 1 of Algorithm 4.3. In Line 2–5, we successively add d and c to S because $source(d)^+ \cap S = \{e\} \cap \{e\} \neq \emptyset$ and $source(c)^+ \cap (S \cup \{d\}) = \{b,d\} \cap \{d,e\} \neq \emptyset$. Afterwards, assume that d is added first to U in Line 7, and that selecting $\{b,c\} \in EB_{\Pi_2}(\{d\})$ in Line 10 leads to $U = \{d\} \cup (\{b,c\} \cap (scc(d) \cap S)) = \{c,d\}$. When investigating $\{a\} \in EB_{\Pi_2}(\{c,d\})$ and again $\{b,c\} \in EB_{\Pi_2}(\{d\})$ in the next two iterations of the loop in Line 8–17, we set $source(c)$ to $\{a\}$ and $source(d)$ to $\{b,c\}$, while obtaining $U = \emptyset$ and $S = \{e\}$. Since $S \neq \emptyset$, another iteration of the loop in Line 6–17 adds e to U and then removes it from U and S along with setting $source(e)$ to $\{c,d\}$. Given $U = S = \emptyset$, we get the empty unfounded set (again surrounded by a box) as result.*

At decision level 3*, unfounded set checking is without effect because, as shown in Table 4.3, no rule body and, in particular, no source pointer is falsified. However, at decision level* 4*, we have that* $source(d) = \{b, c\} \in \mathbf{A}^{\boldsymbol{F}}$*, and thus we get* $S = \{d\}$ *in Line 1 of Algorithm 4.3. In an iteration of the loop in Line 2–5, we further add* e *to* S *because* $source(e)^+ \cap S = \{c, d\} \cap \{d\} \neq \emptyset$*, while* c *stays unaffected in view of* $source(c) = \{a\} \notin \mathbf{A}^{\boldsymbol{F}}$*. After adding* e *to* U *in Line 7,* U *is further extended to* $\{d, e\}$ *in Line 16, given that* $\{c, d\} \in EB_{\Pi_2}(\{e\})$ *and* $\{c, d\} \cap (scc(e) \cap S) = \{d\}$*. We have now obtained* $U = \{d, e\}$*, and it holds that* $EB_{\Pi_2}(\{d, e\}) = \{\{b, c\}, \{b, not\ a\}\} \subseteq \mathbf{A}^{\boldsymbol{F}}$*. That is, the termination condition in Line 9 applies, and* UNFOUNDEDSET$(\Pi_2, \mathbf{A})$ *returns the (non-empty) unfounded set* $\{d, e\}$*.*

To conclude the example, in Table 4.3, we observe that adding the loop nogood $\{\boldsymbol{T}d, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$ *to* ∇ *leads to a conflict at decision level* 4*. After backjumping to decision level* 2*,* NOGOODPROPAGATION *encounters a conflict before invoking* UNFOUNDEDSET$(\Pi_2, \mathbf{A})$*. Hence,* UNFOUNDEDSET$(\Pi_2, \mathbf{A})$ *is only queried again w.r.t. the total assignment* $\mathbf{A}$ *derived by unit propagation after returning to decision level* 1*. In view of* $source(c) = \{a\} \in \mathbf{A}^{\boldsymbol{F}}$*, this final invocation (not shown in Table 4.4) makes us reset source pointers as follows:* $source(e) = \{b, not\ a\}$*,* $source(d) = \{e\}$*, and* $source(c) = \{b, d\}$ *(like at decision level* 0*). As this yields only the empty unfounded set (of non-false atoms),* NOGOODPROPAGATION *terminates without conflict, and* CDNL-ASP(Π_2) *returns the answer set* $\{b, c, d, e\}$ *of* Π_2*.*

4.3.4 Conflict Analysis

The purpose of the subroutine for conflict analysis is to determine an asserting nogood, so that some entry is unit-resulting after backjumping. To this end, a violated nogood $\delta \subseteq \mathbf{A}$ is resolved against antecedents ε of implied entries $\sigma \in \delta$ (i.e., nogoods ε such that $\varepsilon \setminus \mathbf{A}[\sigma] = \{\overline{\sigma}\}$) for obtaining a new violated nogood $(\delta \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\})$. Iterated resolution proceeds in inverse order of assigned entries, resolving first over the entry $\sigma \in \delta$ assigned last in $\mathbf{A}$ (i.e., $\delta \setminus \mathbf{A}[\sigma] = \{\sigma\}$), and stops as soon as δ contains exactly one entry, called *Unique Implication Point* (UIP) [179], assigned at the decision level where the conflict is encountered. The success of this approach, referred to as *First-UIP* scheme (cf. [219, 56, 178]), has in the area of SAT been proven both empirically [219, 203, 48] and analytically [7, 197]. Despite small peculiarities (discussed below), the First-UIP scheme can be applied unaltered in conflict-driven ASP solving. However, identifying antecedents of implied entries is less straightforward than with clauses. For instance, note that our subroutine for propagation in Algorithm 4.2 records a priori implicit loop nogoods from Λ_Π to make sure that every implied entry has some antecedent in $\Delta_\Pi \cup \nabla$.

Conflict resolution according to the First-UIP scheme is performed by CONFLICT-ANALYSIS, shown in Algorithm 4.4. In fact, the loop in Line 1–7 proceeds by resolving over the entry σ of the violated nogood δ assigned last in $\mathbf{A}$ (given that $\delta \setminus \mathbf{A}[\sigma] = \{\sigma\}$ is required in Line 2) until the *assertion level* [41], that is, the greatest level $dlevel(\rho)$ associated with entries $\rho \in \delta \setminus \{\sigma\}$, is different from (and actually smaller than) $dlevel(\sigma)$. If so, the nogood δ and the assertion level k (determined in Line 3) are returned in Line 7; since $\delta \subseteq \mathbf{A}$, we have that $\overline{\sigma}$ is unit-resulting for δ after backjumping to decision level k. Otherwise, if $k = dlevel(\sigma)$, σ is an implied entry, so that some antecedent $\varepsilon \in \Delta_\Pi \cup \nabla$ of σ can be chosen in Line 5 and used for resolution against δ in Line 6. Note that there may be several antecedents of σ in $\Delta_\Pi \cup \nabla$, and thus the choice of ε in Line 5 is, in general, non-deterministic (cf. [53]). Regarding the termination of Algorithm 4.4, note

Algorithm 4.4: CONFLICTANALYSIS

Input : A non-empty violated nogood δ, a normal program Π, a set ∇ of nogoods, and an ordered assignment $\mathbf{A}$.
Output : A derived nogood and a decision level.

```
1 loop
2     let σ ∈ δ such that δ \ A[σ] = {σ} in
3         k := max({dlevel(ρ) | ρ ∈ δ \ {σ}} ∪ {0})
4         if k = dlevel(σ) then
5             let ε ∈ Δ_Π ∪ ∇ such that ε \ A[σ] = {σ̄} in
6                 δ := (δ \ {σ}) ∪ (ε \ {σ̄})                    // resolution
7         else return (δ, k)
```

δ	ε
$\{\boxed{\boldsymbol{T}\{not\ a\}}, \underline{\boldsymbol{T}a}\}$	$\{\underline{\boldsymbol{T}b}, \boxed{\boldsymbol{F}\{not\ a\}}\}$
$\{\boxed{\boldsymbol{T}a}, \underline{\boldsymbol{T}b}\}$	$\{\underline{\boldsymbol{F}\{b, not\ a\}}, \underline{\boldsymbol{T}b}, \boxed{\boldsymbol{F}a}\}$
$\{\boxed{\boldsymbol{T}b}, \underline{\boldsymbol{F}\{b, not\ a\}}\}$	$\{\underline{\boldsymbol{T}\{b, c\}}, \boxed{\boldsymbol{F}b}\}$
$\{\boxed{\boldsymbol{T}\{b, c\}}, \underline{\boldsymbol{F}\{b, not\ a\}}\}$	$\{\boldsymbol{T}d, \boxed{\boldsymbol{F}\{b, c\}}, \underline{\boldsymbol{F}\{b, not\ a\}}\}$
$\{\boxed{\boldsymbol{F}\{b, not\ a\}}, \boldsymbol{T}d\}$	

Table 4.5: Run of CONFLICTANALYSIS($\{\boldsymbol{T}\{not\ a\}, \boldsymbol{T}a\}, \Pi_2, \nabla, \mathbf{A}$) at decision level 2.

that a decision entry σ_d (cf. Algorithm 4.1) is the first entry in $\mathbf{A}$ at its (positive) level $dlevel(\sigma_d)$, and σ_d is also the only entry at $dlevel(\sigma_d)$ that is not implied. Given that CONFLICTANALYSIS is only applied to nogoods violated at decision levels beyond 0, all conflict resolution steps are well-defined and stop at latest at a decision entry σ_d. However, resolving up to σ_d can be regarded as the worst case, given that the First-UIP scheme aims at few resolution steps to obtain a nogood that is "close" to a conflict at hand.

Example 4.9. *To illustrate Algorithm 4.4, let us inspect the resolution steps shown in Table 4.5. They are applied when resolving the violated nogood $\{\boldsymbol{T}\{not\ a\}, \boldsymbol{T}a\}$ against the antecedents shown in Table 4.3, upon analyzing the conflict encountered at decision level 2 in the computation of* CDNL-ASP(Π_2) *described in Example 4.5. The entry σ of δ assigned last in $\mathbf{A}$ as well as its complement $\overline{\sigma}$ in an antecedent ε of σ are surrounded by a box in Table 4.5, and further entries assigned at decision level 2 are underlined. The result of iterated resolution, $\{\boldsymbol{F}\{b, not\ a\}, \boldsymbol{T}d\}$, contains $\boldsymbol{F}\{b, not\ a\}$ as the single entry assigned at decision level 2, while $\boldsymbol{T}d$ has been assigned at the assertion level 1. In this example, the (first) UIP, $\boldsymbol{F}\{b, not\ a\}$, happens to be the decision entry at level 2.*

In general, a first UIP is not necessarily a decision entry, which can, for instance, be observed on the UIP $\boldsymbol{F}\{b, c\}$ in the asserting nogood $\{\boldsymbol{T}d, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$ returned by CONFLICTANALYSIS at decision level 4 in Example 4.5. Also recall that $\{\boldsymbol{T}d, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{b, not\ a\}\}$ served as the starting point for CONFLICTANALYSIS, containing a (first) UIP without requiring any resolution step. This phenomenon is due to

"unidirectional" propagation of loop nogoods, given that unfounded set checks (cf. Algorithm 4.3) merely identify unfounded atoms, but not rule bodies that must necessarily hold for (non-circularly) supporting some true atom. In Example 4.5, the fact that $\boldsymbol{T}\{b, c\}$ is required from decision level 2 on is only recognized at level 4, where assigning $\boldsymbol{F}\{b, c\}$ leads to a conflict. In view of this, Algorithm 4.3 can be understood as a checking routine guaranteeing the soundness of CDNL-ASP, while its inference capabilities do not match (full) unit propagation on loop nogoods. Similar observations have already been made in [111, 110], but more powerful yet efficient reasoning mechanisms for unfounded set handling seem to be difficult to develop; for instance, the approach suggested in [33, 34] is computationally too complex (quadratic) to be beneficial in practice.

Despite of the fact that conflict resolution in ASP can be done in the same fashion as in SAT, the input format of logic programs makes it less predetermined. For one, the completion nogoods in Δ_Π contain rule bodies as structural variables for the sake of a compact representation. For another, the number of (relevant) inherent loop nogoods in Λ_Π may be exponential [164]. Fortunately, the satisfaction of Λ_Π can be checked in linear time (e.g., via Algorithm 4.3), so that an explicit representation of its elements is not required. However, NOGOODPROPAGATION (cf. Algorithm 4.2) records loop nogoods from Λ_Π that are antecedents to make them easily accessible in CONFLICTANALYSIS.

Alternatives in the representation of constraints induced by a logic program become apparent when considering traditional ASP solvers, such as *dlv* [158] and *smodels* [209], where assignments are (logically) identified with interpretations over atoms. In order to augment *smodels* with conflict-driven learning, *smodels*$_{cc}$ [218] pursues an algorithmic approach to extract antecedents (over atoms) relative to *smodels*' propagation rules (described in Section 3.2.2). In our setting, one may restrict heuristic decisions in Line 14 of Algorithm 4.1 to mimic an "atom-only" approach, in which truth values of bodies are determined by their literals. However, if CONFLICTANALYSIS remains unaltered, its (asserting) nogoods may still enable unit propagation to derive the falsity of rule bodies without (known) false body literals (or associated false head atoms), a state that cannot occur with atom-based approaches. To ultimately avoid such states, one would need to unconditionally eliminate entries over bodies from conflict nogoods by resolution against their antecedents, which is possible when heuristic decisions are restricted to atoms. This idea comes close to the conflict-driven learning technique of *smodels*$_{cc}$, breaking derivations relying on bodies down to their contained literals. While such "body elimination" admits the integration of conflict-driven learning into atom-based approaches, it may still go along with exponentially increased (best-case) complexity (cf. Section 3.4.1), which is independent of and thus irreparable by conflict-driven learning [110].

4.3.5 Soundness and Completeness

In the following, we elaborate upon the formal properties of the algorithms presented in the previous sections. Generally speaking, soundness w.r.t. the decision problem of answer set existence is obtained from the fact that the subroutines NOGOODPROPAGATION and CONFLICTANALYSIS exploit and possibly tighten available knowledge, but do not draw incorrect conclusions. In the course of this, UNFOUNDEDSET performs a sufficient amount of work to distinguish answer sets from (inadmissible) circularly supported models. The completeness of CDNL-ASP follows from the observation that its subroutines cannot loop infinitely along with the fact that conflict-driven assertions relocate variables to smaller decision levels than before, which guarantees termination (cf. [220, 203]).

To begin with, we consider crucial properties of UNFOUNDEDSET in Algorithm 4.3. First, we have that (positive) dependencies through source pointers are inherently acyclic.

Lemma 4.5. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment that is body-saturated for* Π.

If UNFOUNDEDSET$(\Pi, \mathbf{A})$ *is invoked on a valid source pointer configuration, then we have that the source pointer configuration remains valid throughout the execution of* UNFOUNDEDSET$(\Pi, \mathbf{A})$.

This property holds because potential non-circular supports for atoms in B^+ must already be established before a source pointer can be set to a body B in Line 13 of Algorithm 4.3, which precludes the introduction of cycles. In particular, the atoms of B^+ belonging to an investigated strongly connected component of $DG(\Pi)$ must not be in the scope S, containing potentially unfounded atoms. In fact, the following result shows that all "interesting" unfounded sets, namely, unfounded loops, are part of S; conversely, atoms outside S cannot belong to an unfounded loop.

Lemma 4.6. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment that is atom-saturated for* Π.

If UNFOUNDEDSET$(\Pi, \mathbf{A})$ *is invoked on a valid source pointer configuration, then we have that every unfounded set* $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ *of* Π *w.r.t.* $\mathbf{A}$ *such that all* $p \in U$ *belong to the same strongly connected component of* $DG(\Pi)$ *is contained in* S *whenever Line 6 of Algorithm 4.3 is entered.*

The previous lemmas along with Corollary 2.10 on Page 15 can now be combined to, essentially, establish the completeness of Algorithm 4.3.[5]

Theorem 4.7. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment that is both atom- and body-saturated for* Π.

If UNFOUNDEDSET$(\Pi, \mathbf{A})$ *is invoked on a valid source pointer configuration, then we have that* UNFOUNDEDSET$(\Pi, \mathbf{A})$ *returns an unfounded set* $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ *of* Π *w.r.t.* $\mathbf{A}$*, where* $U = \emptyset$ *iff there is no unfounded set* U' *of* Π *w.r.t.* $\mathbf{A}$ *such that* $U' \not\subseteq \mathbf{A}^{\boldsymbol{F}}$.

After considering unfounded set detection, we now turn to NOGOODPROPAGATION in Algorithm 4.2. The next lemma is straightforward yet helpful, as it assures the prerequisites of demand-driven unfounded set checking, mainly focusing on unfounded loops.

Lemma 4.8. *Let* Π *be a normal program,* ∇' *a set of nogoods, and* $\mathbf{A}'$ *an assignment.*

Then, we have that $\mathbf{A}$ *is both atom- and body-saturated for* Π *whenever Line 10 of Algorithm 4.2 is entered in an execution of* NOGOODPROPAGATION$(\Pi, \nabla', \mathbf{A}')$.

The following properties are essential for CONFLICTANALYSIS to be well-defined as well as the soundness and completeness of CDNL-ASP.

Lemma 4.9. *Let* Π *be a normal program,* ∇' *a set of nogoods, and* $\mathbf{A}'$ *an ordered assignment.*

If NOGOODPROPAGATION$(\Pi, \nabla', \mathbf{A}')$ *is invoked on a valid source pointer configuration, then we have that* NOGOODPROPAGATION$(\Pi, \nabla', \mathbf{A}')$ *returns a pair* $(\mathbf{A}, \nabla)$ *such that*

[5] Soundness, viz., the property that every set U returned by UNFOUNDEDSET is indeed unfounded, is obvious in view of the test in Line 9 of Algorithm 4.3 and the fact that $\emptyset$, which can be returned in Line 18, is trivially unfounded.

1. $\nabla' \subseteq \nabla \subseteq \nabla' \cup \Lambda_\Pi$;

2. $\mathbf{A}$ *is an ordered assignment such that* $\mathbf{A}' \subseteq \mathbf{A}$ *and every* $\sigma \in \mathbf{A} \setminus \mathbf{A}'$ *is implied by* $\Delta_\Pi \cup \nabla$ *w.r.t.* $\mathbf{A}$;

3. $\delta \subseteq \mathbf{A}$ *for some* $\delta \in \Delta_\Pi \cup \nabla$ *if* $\varepsilon \subseteq \mathbf{A}$ *for some* $\varepsilon \in \Lambda_\Pi$.

The first item expresses that only loop nogoods can possibly be added by NOGOOD-PROPAGATION, viz., in Line 15 of Algorithm 4.2 (provided that Π is non-tight). In view of Theorem 4.4 on Page 59, this makes sure that the recorded nogoods do not eliminate any answer set of Π. The second item brings forward that any entry derived by NOGOOD-PROPAGATION has some antecedent, which can (later on) be used for conflict resolution. Finally, the third item exploits Theorem 4.7 and Lemma 4.8 to establish that violations of (loop) nogoods cannot stay undetected.

Regarding CONFLICTANALYSIS in Algorithm 4.4, the next lemma states that its derived nogoods are asserting and entailed by the nogoods that are already given.

Lemma 4.10. *Let* Π *be a normal program,* ∇ *a set of nogoods,* $\mathbf{A}$ *an ordered assignment, and* $\delta' \subseteq \mathbf{A}$ *such that* $m = \max(\{dlevel(\rho) \mid \rho \in \delta'\} \cup \{0\}) \neq 0$.

If m *is implied by* $\Delta_\Pi \cup \nabla$ *w.r.t.* $\mathbf{A}$*, then we have that* CONFLICT-ANALYSIS$(\delta', \Pi, \nabla, \mathbf{A})$ *returns a pair* (δ, k) *such that*

1. $\delta \subseteq \mathbf{A}$;

2. $|\{\sigma \in \delta \mid 0 \leq k < dlevel(\sigma)\}| = 1$;

3. $\delta \not\subseteq \mathbf{B}$ *for any solution* $\mathbf{B}$ *for* $\Delta_\Pi \cup \nabla \cup \{\delta'\}$.

The above prerequisites regarding ∇, $\mathbf{A}$, and δ' stipulate the existence of antecedents for all but the first entry assigned at the decision level $m > 0$. These conditions are guaranteed by CDNL-ASP, as it introduces a new decision level in Line 15 of Algorithm 4.1 before assigning a decision entry (without antecedent) in Line 16, and as conflicts are only analyzed if they are encountered beyond decision level 0.

After inspecting the subroutines of CDNL-ASP, important invariants of assignments and nogoods generated by CDNL-ASP can be summarized as follows.

Lemma 4.11. *Let* Π *be a normal program.*

Then, we have that the following holds whenever Line 5 of Algorithm 4.1 is entered in an execution of CDNL-ASP(Π)*:*

1. ∇ *is a set of nogoods such that* $\delta \not\subseteq \mathbf{B}$ *for every* $\delta \in \nabla$ *and any solution* $\mathbf{B}$ *for* $\Delta_\Pi \cup \Lambda_\Pi$;

2. $\mathbf{A}$ *is an ordered assignment such that all decision levels are implied by* $\Delta_\Pi \cup \nabla$ *w.r.t.* $\mathbf{A}$.

Given that only the (implied) entries belonging to the current assignment $\mathbf{A}$ require antecedents for the second invariant to hold, dynamic nogoods in ∇ that are not antecedents may optionally be deleted. This yields polynomial space complexity of CDNL-ASP because the number of (required) antecedents is bounded by the number of entries,

viz., by $|atom(\Pi) \cup body(\Pi)|$.[6] In practice, nogood deletion (cf. [127, 56]) is an important technique preventing conflict-driven learning solvers from blowing up in space.

In order to capture the notion of completeness also in the case of unsatisfiability, i.e., for logic programs having no answer set, we introduce the following terminology: an algorithm is called *terminating* if it finishes in finitely many steps. By combining the previous results, we can now establish the soundness and completeness of CDNL-ASP.

Theorem 4.12. *Let* Π *be a normal program.*

Then, we have that CDNL-ASP(Π) *is terminating, and it returns an answer set of* Π *iff* Π *has some answer set.*

Soundness w.r.t. the decision problem of answer set existence follows from the observations made above, namely, that violations of (loop) nogoods are detected and that nogoods added by NOGOODPROPAGATION or derived by CONFLICTANALYSIS are entailed. The completeness of CDNL-ASP, i.e., the fact that it is a decision procedure, is due to its termination. Notably, the arguments for the termination of CDCL (cf. [220, 203]) also apply to CDNL-ASP, given that both search procedures make use of conflict-driven assertions, which exclude repetitions of assignments.

4.4 Enumeration Algorithms

The CDNL-ASP algorithm presented in Section 4.3.1 solves the decision problem of answer set existence for normal programs, thereby, making use of state-of-the-art lookback techniques like conflict-driven learning and backjumping according to the First-UIP scheme. Unlike with DPLL-style procedures performing systematic backtracking, the transition from computing a single solution to computing multiple or all of them is non-trivial in the context of state-of-the-art lookback techniques. Some methods developed in the area of SAT utilize "blocking clauses" (cf. [151, 146, 184]) to remember previously found solutions for excluding their repetition; such approaches incur significant blow-up in space.[7] This could be avoided by deleting non-asserting blocking clauses, thereby, still guaranteeing termination but risking the repetition of solutions [155]. Further methods are devised particularly for #SAT (see [128] for an overview), the problem of counting the models of a propositional theory; the ones among them that include conflict-driven learning and backjumping according to the First-UIP scheme incur significant computational overhead a priori. In contrast to the aforementioned proposals, our goal is to devise algorithms for the enumeration of answer sets that

1. harness conflict-driven learning and backjumping according to the First-UIP scheme,

[6] The total number and the individual size of antecedents in ∇ that must not be deleted (as some decision level would not be implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$ otherwise) are both linearly bounded by $|atom(\Pi) \cup body(\Pi)|$. Hence, the space complexity of CDNL-ASP is at most quadratic in the size of Π. Similar considerations are applicable to CDCL w.r.t. the number of propositional variables occurring in a set of clauses. However, the author is unaware of any precise estimation of the space needed to store (required) antecedents, comparing their size to the size of input clauses in terms of occurrences of propositional variables.

[7] The enumeration approach of *relsat* [17, 16] works without blocking clauses, but *relsat*'s lookback techniques rely on the "Last-UIP" scheme (cf. [219]). The "Important First Decision Procedure" proposed in [130] also avoids the use of blocking clauses, but it interferes with decision heuristics and backjumping; we further compare our approach to it in Section 4.6.

2. do not incur computational overhead a priori, that is, before any answer set is found,

3. are applicable to the counting problem by determining an exact number of (distinct) solutions, and

4. run in polynomial space, provided that unrequired conflict and loop nogoods are deleted.

Although it does not comply with our objective, in Section 4.4.1, we first describe a straightforward enumeration algorithm based on solution recording. It serves as a starting point for the development of more sophisticated algorithms applying dedicated backtracking schemes instead of space-consuming solution recording. Such algorithms, aiming at the enumeration of entire solutions and projections to a subset of variables, respectively, are presented in Section 4.4.2 and 4.4.3. Finally, Section 4.4.4 outlines the derivation of soundness and completeness results for the devised enumeration algorithms.

4.4.1 Solution Recording

The main enumeration algorithm based on solution recording, CDNL-RECORDING, is shown in Algorithm 4.5. It extends CDNL-ASP in Algorithm 4.1 on Page 62 by taking as second argument a (maximum) number s of answer sets to enumerate; all answer sets of a normal program Π can be computed by letting s be 0. CDNL-ASP and CDNL-RECORDING use the same subroutines and are in most parts identical, yet their behaviors differ when an answer set is found (Line 11–12 in Algorithm 4.1 and Line 11–15 in Algorithm 4.5). Unlike CDNL-ASP, which terminates immediately after finding an answer set, CDNL-RECORDING prints the answer set in Line 12, decrements the number s of answer sets to enumerate in Line 13, and checks in Line 14 whether more answer sets are requested. If so, in Line 15, it adds the set of all entries $\sigma_p \in \mathbf{A}$ over $atom(\Pi)$ to ∇ for prohibiting a repetition of the corresponding answer set in the sequel. In fact, when such a nogood is added to ∇ in Line 15, it is violated by $\mathbf{A}$ and thus triggers a conflict in the next iteration of the loop in Line 4–19.

Unlike conflict and loop nogoods, which are entailed by the input and may optionally be deleted when they are no longer required as antecedents (cf. Section 4.3.5), the nogoods added to ∇ in Line 15 must not be deleted. Hence, Algorithm 4.5 is prone to blow up in space. We note that there are approaches to represent the set of enumerated solutions more compactly (cf. [146, 184]), but we do not make use of them because it is not our goal to improve persistent solution recording. Rather, we aim at enumeration algorithms that cope without such recording, presented here for comparison only. We also do not exploit the "antichain property" satisfied by the answer sets of a normal program (cf. [120]) and still include entries of the form $\boldsymbol{F}p$ for $p \in atom(\Pi)$ in a nogood added to ∇ in Line 15. In fact, the (antichain) property that two distinct answer sets are incomparable w.r.t. inclusion does not hold in syntactically richer settings, e.g., it is not guaranteed by the answer sets of a nested program [165].

Example 4.10. *Consider the following normal program:*

$$\Pi_{11} = \left\{ \begin{array}{lll} r_1 : a \leftarrow x & r_3 : b \leftarrow x & r_5 : c \leftarrow x, not\ z \\ r_2 : a \leftarrow not\ x & r_4 : b \leftarrow not\ c & r_6 : c \leftarrow not\ b \\ r_7 : x \leftarrow b, c & r_9 : y \leftarrow x, not\ b & r_{11} : z \leftarrow x, not\ c \\ r_8 : x \leftarrow not\ y, not\ z & r_{10} : y \leftarrow not\ x, not\ z & r_{12} : z \leftarrow not\ x, not\ y \end{array} \right\}$$

Algorithm 4.5: CDNL-RECORDING

Input : A normal program Π and a number s of answer sets to enumerate.

```
1  A := ∅                                          // ordered assignment over atom(Π) ∪ body(Π)
2  ∇ := ∅                                          // set of recorded nogoods
3  dl := 0                                         // decision level
4  loop
5      (A, ∇) := NogoodPropagation(Π, ∇, A)
6      if ε ⊆ A for some ε ∈ Δ_Π ∪ ∇ then          // conflict
7          if max({dlevel(σ) | σ ∈ ε} ∪ {0}) = 0 then exit
8          (δ, dl) := ConflictAnalysis(ε, Π, ∇, A)
9          ∇ := ∇ ∪ {δ}                          // (temporarily) record conflict nogood
10         A := A \ {σ ∈ A | dl < dlevel(σ)}      // backjumping
11     else if A^T ∪ A^F = atom(Π) ∪ body(Π) then  // answer set
12         print A^T ∩ atom(Π)
13         s := s − 1
14         if s = 0 then exit
15         ∇ := ∇ ∪ {{σ_p ∈ A | var(σ_p) ∈ atom(Π)}}   // (persistently) record solution
16     else
17         σ_d := Select(Π, ∇, A)                  // decision
18         dlevel(σ_d) := dl := dl + 1
19         A := A ∘ σ_d
```

Let us examine a computation of all answer sets of Π_{11} *with* CDNL-RECORDING$(\Pi_{11}, 0)$*, whose main steps are shown in Table 4.6. As in Table 4.3, the columns provide the value of* dl*, viz., the current decision level, and the line of Algorithm 4.5 at which particular contents of* $\mathbf{A}$ *and/or some nogood* δ *are inspected. The info column displays additional information about printed answer sets, nogoods, and assertion levels, respectively. The entries inserted into* $\mathbf{A}$ *in Line 19 of Algorithm 4.5 are decision entries. Unlike them, each entry inserted into* $\mathbf{A}$ *in Line 5, that is, in an execution of* NOGOODPROPAGATION*, is unit-resulting for some nogood* $\delta \in \Delta_{\Pi_{11}} \cup \nabla$*. We merely indicate by "..." that some entries are derived by unit propagation but, for brevity, omit listing individual entries. However, Table 4.6 displays successes of the test for a violated nogood performed in Line 6, and it provides the nogood* δ *to be recorded in* ∇ *along with the assertion level* dl *(as info) as they are returned by* CONFLICTANALYSIS *when invoked in Line 8. In addition, the info column shows answer sets (underlined) when they are printed in Line 12. In the following, we describe the macroscopic steps of* CDNL-RECORDING$(\Pi_{11}, 0)$*; an unabridged trace is provided in Appendix A.1.*

Assume that, at the start of the computation, $\boldsymbol{T}y$*,* $\boldsymbol{T}a$*, and* $\boldsymbol{T}x$ *are picked as decision entries at levels* 1*,* 2*, and* 3*, respectively. This leads to the violation of the completion nogood* $\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$ *by unit propagation, and the derived conflict nogood,* $\{\boldsymbol{T}x, \boldsymbol{T}y\}$*, is asserting at decision level* 1*. Similar to* CDNL-ASP*, this makes* CDNL-RECORDING *backjump to decision level* 1*, where* $\boldsymbol{F}x$ *and further entries are afterwards derived by unit propagation. As a consequence of picking* $\boldsymbol{T}b$ *as the new decision entry at level* 2*, the first solution is obtained. Its true entries over atoms provide us with an answer set of* Π_{11}*,* $\{y, a, b\}$*, which is printed in Line 12 of Algorithm 4.5. The associated set of entries over atoms is* $\{\boldsymbol{T}y, \boldsymbol{F}x, \boldsymbol{F}z, \boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}c\}$*. Since* s *is decremented to* -1 *in Line 13,* CDNL-

dl	**A**	δ	Info	Line
1	$\boldsymbol{T}y$			19
	...			5
2	$\boldsymbol{T}a$			19
3	$\boldsymbol{T}x$			19
	...			5
		$\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	6
		$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	dl=1	8
1	$\boldsymbol{T}y$			
	...			5
2	$\boldsymbol{T}b$			19
	...			5
			$\underline{\{y, a, b\}}$	12
		$\{\boldsymbol{T}y, \boldsymbol{F}x, \boldsymbol{F}z, \boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}c\}$	∇	6
		$\{\boldsymbol{T}y, \boldsymbol{F}x, \boldsymbol{F}z, \boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}\{x\}\}$	dl=1	8
1	$\boldsymbol{T}y$			
	...			5
			$\underline{\{y, a, c\}}$	12
		$\{\boldsymbol{T}y, \boldsymbol{F}x, \boldsymbol{F}z, \boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{T}c\}$	∇	6
		$\{\boldsymbol{T}y\}$	dl=**0**	8
0	$\boldsymbol{F}y$	$\{\boldsymbol{T}y\}$	∇	5
	...			5
1	$\boldsymbol{T}b$			19
	...			5
2	$\boldsymbol{T}c$			19
	...			5
			$\underline{\{x, a, b, c\}}$	12
		$\{\boldsymbol{F}y, \boldsymbol{T}b, \boldsymbol{T}c, \boldsymbol{T}a, \boldsymbol{T}x, \boldsymbol{F}z\}$	∇	6
		$\{\boldsymbol{F}y, \boldsymbol{T}b, \boldsymbol{T}c, \boldsymbol{F}\{not\ b\}\}$	dl=1	8
0	$\boldsymbol{F}y$	$\{\boldsymbol{T}y\}$	∇	
	...			
1	$\boldsymbol{T}b$			
	...			5
2	$\boldsymbol{F}a$			19
	...			5
		$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	6
		$\{\boldsymbol{F}a\}$	dl=**0**	8
0	$\boldsymbol{F}y$	$\{\boldsymbol{T}y\}$	∇	
	...			5
1	$\boldsymbol{F}z$			19
	...			5
		$\{\boldsymbol{F}y, \boldsymbol{T}b, \boldsymbol{T}c, \boldsymbol{T}a, \boldsymbol{T}x, \boldsymbol{F}z\}$	∇	6
		$\{\boldsymbol{F}y, \boldsymbol{T}a, \boldsymbol{F}z\}$	dl=**0**	8
0	$\boldsymbol{F}y$	$\{\boldsymbol{T}y\}$	∇	
	...			5
1	$\boldsymbol{T}b$			19
	...			5
			$\underline{\{z, a, b\}}$	12
		$\{\boldsymbol{F}y, \boldsymbol{T}a, \boldsymbol{T}z, \boldsymbol{T}b, \boldsymbol{F}c, \boldsymbol{F}x\}$	∇	6
		$\{\boldsymbol{F}y, \boldsymbol{T}a, \boldsymbol{T}z, \boldsymbol{T}b, \boldsymbol{F}\{not\ y, not\ z\}, \boldsymbol{F}\{x, not\ z\}\}$	dl=**0**	8
0	$\boldsymbol{F}y$	$\{\boldsymbol{T}y\}$	∇	
	...			5
			$\underline{\{z, a, c\}}$	12
		$\{\boldsymbol{F}y, \boldsymbol{T}a, \boldsymbol{T}z, \boldsymbol{F}b, \boldsymbol{T}c, \boldsymbol{F}x\}$	∇	6

Table 4.6: Main steps in a computation of all answer sets with CDNL-RECORDING($\Pi_{11}, 0$).

RECORDING *does not exit in Line 14 and proceeds by recording the aforementioned set of entries over atoms persistently in* ∇ *as a nogood. This triggers a conflict in the next iteration of the loop in Line 4–19, so that* CONFLICTANALYSIS *in Line 8 determines an asserting nogood along with an assertion level, as in the case of a nogood violated upon propagation. In our example, the asserting nogood* $\{\boldsymbol{T}y, \boldsymbol{F}x, \boldsymbol{F}z, \boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}\{x\}\}$ *leads to a second solution after returning to decision level* 1 *and performing unit propagation. The corresponding answer set of* Π_{11}*,* $\{y, a, c\}$*, is then excluded by persistently recording* $\{\boldsymbol{T}y, \boldsymbol{F}x, \boldsymbol{F}z, \boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{T}c\}$ *in* ∇*. As a consequence,* CONFLICTANALYSIS *in Line 8 yields the asserting nogood* $\{\boldsymbol{T}y\}$*, whose assertion level is* 0*. This shows that all solutions for* $\Delta_{\Pi_{11}} \cup \Lambda_{\Pi_{11}}$ *containing* $\boldsymbol{T}y$ *(that is, all answer sets of* Π_{11} *containing* y*) have been enumerated.*

After backjumping, the fact that $\boldsymbol{F}y$ *is derived by unit propagation at decision level* 0 *is highlighted by writing* $\mathbf{0}$ *in Table 4.6. Note that the violation of a nogood such that all entries are assigned at decision level* 0 *is the termination condition of Algorithm 4.5, which must eventually apply in Line 7 if* 0 *is provided for the second argument* s*; hence, entries derived at decision level* 0 *are of particular significance. To summarize the remaining part of the computation, three more solutions, corresponding to the answer sets* $\{x, a, b, c\}$*,* $\{z, a, b\}$*, and* $\{z, a, c\}$ *of* Π_{11}*, are obtained and excluded by persistently recording associated nogoods in* ∇ *before the last such nogood is violated at decision level* 0*. In the course of this, it is noteworthy to mention that one of the recorded conflict nogoods,* $\{\boldsymbol{F}a\}$*, implies the entry* $\boldsymbol{T}a$ *after backjumping to decision level* 0*. The resulting partial assignment has already been contained in the previously enumerated solution with* $\{\boldsymbol{F}y, \boldsymbol{T}b, \boldsymbol{T}c, \boldsymbol{T}a, \boldsymbol{T}x, \boldsymbol{F}z\}$ *as its set of entries over atoms. Due to persistently recording this set of entries as a nogood, a repetition of the previous solution is suppressed, as it can be observed on the second last conflict encountered at decision level* 1*. In fact, by a posteriori turning solutions into nogoods,* CDNL-RECORDING *successively strengthens* $\Delta_{\Pi_{11}} \cup \Lambda_{\Pi_{11}}$ *until unsatisfiability certifies that no unenumerated solution is left.*

4.4.2 Solution Enumeration

After presenting a straightforward yet space-exploding approach to enumeration for comparison, we now turn to our actual objective to devise an enumeration algorithm that complies with the design goals postulated at the beginning of Section 4.4.

Our main algorithm for enumeration via a dedicated backtracking scheme, CDNL-ENUMERATION, is shown in Algorithm 4.6. Its arguments, a normal program Π and a (maximum) number s of answer sets to enumerate, are similar to Algorithm 4.5. The additional key ingredient of Algorithm 4.6 is a *backtracking level*, denoted by bl, used in addition to the decision level dl. At any stage of the computation, bl provides the greatest level such that some enumerated solution contains all decision entries assigned at levels up to bl. Since these entries (along with all further entries assigned at decision levels smaller than bl) belong to a solution, any conflict-driven assertion (assigning a unit-resulting entry for an asserting nogood after backjumping) at a decision level smaller than bl must reestablish some entry of an already enumerated solution. To avoid repetitions, we thus have to make sure that backjumping does not retract bl, but only leads to the insertion of a unit-resulting entry (of an already enumerated solution) into $\mathbf{A}$ at the decision level where it is implied. Exactly this is accomplished in Line 11 of Algorithm 4.6 by taking the maximum of the assertion level k, as returned by CONFLICTANALYSIS in Line 9, and bl as the decision level dl to resume from after backjumping in Line 12.

Algorithm 4.6: CDNL-ENUMERATION

Input : A normal program Π and a number s of answer sets to enumerate.

```
1  A := ∅                                          // ordered assignment over atom(Π) ∪ body(Π)
2  ∇ := ∅                                          // set of recorded nogoods
3  bl := dl := 0                                   // (systematic) backtracking and decision level
4  loop
5      (A, ∇) := NogoodPropagation(Π, ∇, A)
6      if ε ⊆ A for some ε ∈ Δ_Π ∪ ∇ then           // conflict
7          if max({dlevel(σ) | σ ∈ ε} ∪ {0}) = 0 then exit
8          if bl < max{dlevel(σ) | σ ∈ ε} then
9              (δ, k) := ConflictAnalysis(ε, Π, ∇, A)
10             ∇ := ∇ ∪ {δ}                        // (temporarily) record conflict nogood
11             dl := max{k, bl}
12             A := A \ {σ ∈ A | dl < dlevel(σ)}   // (bounded) backjumping
13         else
14             σ_d := decision(bl)
15             dlevel(σ̄_d) := bl := dl := bl − 1
16             A := A \ {σ ∈ A | bl < dlevel(σ)}   // backtracking
17             A := A ∘ σ̄_d                        // flipping
18     else if A^T ∪ A^F = atom(Π) ∪ body(Π) then  // answer set
19         print A^T ∩ atom(Π)
20         s := s − 1
21         if s = 0 or dl = 0 then exit
22         σ_d := decision(dl)
23         dlevel(σ̄_d) := bl := dl := dl − 1
24         A := A \ {σ ∈ A | bl < dlevel(σ)}       // backtracking
25         A := A ∘ σ̄_d                            // flipping
26     else
27         σ_d := Select(Π, ∇, A)                  // decision
28         dlevel(σ_d) := dl := dl + 1
29         decision(dl) := σ_d
30         A := A ∘ σ_d
```

Furthermore, before invoking CONFLICTANALYSIS in Line 9, the test in Line 8 checks whether a violated nogood contains some entry assigned at a decision level beyond bl. This is important for guaranteeing conflict resolution to be well-defined because Algorithm 4.6 performs systematic (chronological) backtracking instead of solution recording. In fact, the decision entry σ_d at a level dl is retained in $decision(dl)$ in Line 29 (before assigning σ_d in Line 30), and its complement $\overline{\sigma_d}$ is in Line 17 or 25 assigned at decision level $bl - 1$ or $dl - 1$ upon encountering a conflict at bl[8] (Line 14–17) or a solution at dl (Line 18–25), respectively. In either case, $\overline{\sigma_d}$ is assigned at the new backtracking level bl (set in Line 15 or 23) without being implied by any nogood in $\Delta_\Pi \cup \nabla$. At levels up to bl, such backtracking (without solution recording) tolerates entries that are neither decisions nor implied (but deliberately flipped in view of search

[8]Since all entries of $\mathbf{A}$ assigned at decision levels smaller than bl belong to a previously enumerated solution that is not excluded by any nogood, $bl > \max\{dlevel(\sigma) \mid \sigma \in \varepsilon\}$ cannot hold and $bl = \max\{dlevel(\sigma) \mid \sigma \in \varepsilon\}$ must be the case if the test in Line 8 of Algorithm 4.6 fails.

space exhaustion). Hence, conflict resolution is not well-defined at decision levels up to bl, and the test in Line 8 precludes undefined behavior.

Compared to CDNL-RECORDING in Algorithm 4.5, where backjumping is always accompanied by a conflict-driven assertion, CDNL-ENUMERATION applies systematic backtracking to proceed after encountering a solution or a conflict resulting from flipped entries. In order to avoid repetitions, CDNL-ENUMERATION also bounds backjumping not to retract flipped decision entries of already enumerated solutions. As it does not rely on persistent solution recording, the backtracking scheme of CDNL-ENUMERATION allows for enumerating all answer sets of a logic program in polynomial space (once printed, answer sets need not be remembered) because only the dynamic nogoods that serve as antecedents must be kept, while all others are "redundant" and may thus be deleted. In fact, CDNL-ENUMERATION preserves the space complexity of CDNL-ASP (in Algorithm 4.1 on Page 62), the basic decision procedure it extends.

Example 4.11. *The main steps in a computation of all answer sets of* Π_{11} *from Example 4.10 with* CDNL-ENUMERATION$(\Pi_{11}, 0)$ *are shown in Table 4.7, which indicates the macroscopic steps of* CDNL-ENUMERATION$(\Pi_{11}, 0)$ *in the same fashion as Table 4.6 for* CDNL-RECORDING$(\Pi_{11}, 0)$. *In fact, we below mainly focus on describing the differences between dedicated backtracking-based enumeration and enumeration via solution recording; an unabridged trace of* CDNL-ENUMERATION$(\Pi_{11}, 0)$ *is provided in Appendix A.2.*

At the start (on the left-hand side of Table 4.7), the computation of CDNL-ENUMERATION *proceeds like* CDNL-RECORDING*, encountering a conflict after picking* $\boldsymbol{T}y$, $\boldsymbol{T}a$, *and* $\boldsymbol{T}x$ *as the first three decision entries and, as a consequence of backjumping to decision level* 1 *and picking* $\boldsymbol{T}b$ *as decision entry at level* 2*, obtaining* $\{y, a, b\}$*, printed in Line 19 of Algorithm 4.6, as the first answer set of* Π_{11}. *Then, the test in Line 21 fails because* s *is decremented to* -1 *in Line 20 and the current value of* dl *is* 2*, so that* CDNL-ENUMERATION *does not exit. Unlike* CDNL-RECORDING*, where solutions are turned into nogoods,* CDNL-ENUMERATION *proceeds in Line 22–25 by setting the backtracking level* bl *to the second greatest decision level of entries in* $\mathbf{A}$*, viz., to* 1*, and by inserting the complement* $\boldsymbol{F}b$ *of the formerly last decision entry* $\boldsymbol{T}b$ *into* $\mathbf{A}$ *after backtracking to* bl*. (The value of* bl *is highlighted by writing* $\mathbf{1}$ *in Table 4.7.) Due to flipping* $\boldsymbol{T}b$ *to* $\boldsymbol{F}b$*, we obtain* $\{y, a, c\}$ *as the second answer set of* Π_{11}. *Given that the current decision level is* 1*, continuing as after the first answer set yields* 0 *as the new backtracking level* bl*, at which the complement* $\boldsymbol{F}y$ *of the former decision entry* $\boldsymbol{T}y$ *is assigned. The same partial assignment has also been generated with* CDNL-RECORDING *(cf. Table 4.6), but there* $\boldsymbol{F}y$ *is implied by a conflict nogood derived from recorded solutions. Unlike this,* CDNL-ENUMERATION *assigns* $\boldsymbol{F}y$ *at decision level* 0 *because its backtracking scheme makes sure that all solutions containing* $\boldsymbol{T}y$ *have been enumerated, so that the value of* y *can be flipped independently of nogoods.*

Given $\boldsymbol{F}y$ *at decision level* 0*, two further decisions on* $\boldsymbol{T}b$ *and* $\boldsymbol{T}c$ *yield* $\{x, a, b, c\}$ *as the third answer set of* Π_{11}. *Then, the backtracking level* bl *is increased to* 1*, and* $\boldsymbol{F}c$ *is assigned at level* 1 *(on the right-hand side of Table 4.7), analogously to the previous two solutions. After picking* $\boldsymbol{F}a$ *as decision entry at level* 2*, we encounter a conflict from which the nogood* $\{\boldsymbol{F}a\}$ *is derived; this tells us that every solution must contain* $\boldsymbol{T}a$*, independently of decisions. In fact,* CONFLICTANALYSIS *in Line 9 of Algorithm 4.6 returns* $\{\boldsymbol{F}a\}$ *along with* $k = 0$*. Since the backtracking level* bl *is* 1*, it is in Line 11 taken as the decision level* dl *to resume from after backjumping. That is, the backjump that would*

dl	**A**	δ	Info	Line
1	$\boldsymbol{T}y$			30
	...			5
2	$\boldsymbol{T}a$			30
3	$\boldsymbol{T}x$			30
	...			5
		$\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	6
		$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	dl=1	9
1	$\boldsymbol{T}y$			
	...			
	$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	∇	5
	...			5
2	$\boldsymbol{T}b$			30
	...			5
			$\underline{\{y, a, b\}}$	19
1	$\boldsymbol{T}y$			
	...			
	$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	∇	
	...			
	$\boldsymbol{F}b$			25
	...			5
			$\underline{\{y, a, c\}}$	19
0	$\boldsymbol{F}y$			25
	...			5
1	$\boldsymbol{T}b$			30
	...			5
2	$\boldsymbol{T}c$			30
	...			5
			$\underline{\{x, a, b, c\}}$	19

dl	**A**	δ	Info	Line
0	$\boldsymbol{F}y$			
	...			
1	$\boldsymbol{T}b$			
	...			
	$\boldsymbol{F}c$			25
	...			5
2	$\boldsymbol{F}a$			30
	...			5
		$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	6
		$\{\boldsymbol{F}a\}$	dl=**1**	9
0	$\boldsymbol{F}y$			
	...			
	$\boldsymbol{T}a$	$\{\boldsymbol{F}a\}$	∇	5
1	$\boldsymbol{T}b$			
	...			
	$\boldsymbol{F}c$			
	...			
2	$\boldsymbol{F}x$			30
	...			5
			$\underline{\{z, a, b\}}$	19
0	$\boldsymbol{F}y$			
	...			
	$\boldsymbol{T}a$	$\{\boldsymbol{F}a\}$	∇	
1	$\boldsymbol{T}b$			
	...			
	$\boldsymbol{F}c$			
	...			
	$\boldsymbol{T}x$			25
	...			5
		$\{\boldsymbol{T}\{not\ y, not\ z\}, \boldsymbol{T}z\}$	$\Delta_{\Pi_{11}}$	6
0	$\boldsymbol{F}y$			
	...			
	$\boldsymbol{T}a$	$\{\boldsymbol{F}a\}$	∇	
	$\boldsymbol{F}b$			17
	...			5
			$\underline{\{z, a, c\}}$	19

Table 4.7: Main steps in a computation of all answer sets with CDNL-ENUMERATION($\Pi_{11}, 0$).

in CDNL-ASP *and* CDNL-RECORDING *lead to assertion level* 0 *is here bounded by the backtracking level. This is because all formerly enumerated solutions have contained* $\boldsymbol{T}a$ *(*a *has been included in the corresponding answer sets), and sacrificing the state information accumulated at backtracking level* 1 *would readmit already enumerated solutions (such as the one corresponding to the answer set* $\{x, a, b, c\}$*). Hence, it is sensible to maintain the entries at the backtracking level, in particular, the flipped former decision entry* $\boldsymbol{F}c$*, while* $\boldsymbol{T}a$ *is still inserted into* $\mathbf{A}$ *at decision level* 0 *(within* NOGOODPROPAGATION*) after returning to backtracking level* 1*.*

In the sequel, picking $\boldsymbol{F}x$ *as decision entry at level* 2 *yields* $\{z, a, b\}$ *as the fourth answer set of* Π_{11}*. Like with previous solutions, we backtrack to level* 1 *and then assign* $\boldsymbol{T}x$*. This however leads to a conflict at the backtracking level. In view of the test in Line 8 of Algorithm 4.6, the conflict is not analyzed, but backtracking in Line 14–17 proceeds in the same fashion as in the case of a solution (Line 22–25). That is, we decrement the backtracking level to* 0 *and further assign the complement* $\boldsymbol{F}b$ *of the former decision entry* $\boldsymbol{T}b$*. Under these prerequisites, we obtain* $\{z, a, c\}$ *as the fifth answer set of* Π_{11}*. Since the current value of* dl *is* 0*, there is no decision entry left to flip, and* CDNL-ENUMERATION *finishes in Line 21.*

The fact that CDNL-ENUMERATION uses a backtracking level to keep track of enumerated as well as yet uninvestigated solutions abolishes the need of recording solutions, so that enumeration can be accomplished in polynomial space. As the backtracking level (partially) suppresses conflict-driven learning (according to the First-UIP scheme) and bounds backjumping (as well as restarts; cf. Section 4.5.2) only after solutions have been obtained, it does neither interfere with search nor incur any computational overhead a priori. Hence, CDNL-ENUMERATION is as effective as CDNL-ASP, the basic decision procedure it extends, on unsatisfiable normal programs (having no answer set). As shown in Section 4.4.4, CDNL-ENUMERATION is complete and does not repeat answer sets, so that it can be applied to count answer sets. We have thus achieved the design goals for enumeration algorithms postulated at the beginning of Section 4.4. To our knowledge, CDNL-ENUMERATION is the first method that jointly complies with all of these design goals.

4.4.3 Solution Projection

After considering the enumeration of entire solutions, we now shift our focus to the problem of enumerating solutions distinct on a certain subset of relevant variables or atoms, respectively. For instance, Π_{11} from Example 4.10 has five answer sets: $\{x, a, b, c\}$, $\{y, a, b\}$, $\{y, a, c\}$, $\{z, a, b\}$, and $\{z, a, c\}$. If we limit the attention to the set $\{a, b, c\}$ as relevant atoms (as it can also be done in solution correspondence checking [58]), we are left with only three distinguishable answer sets: $\{a, b, c\}$, $\{a, b\}$, and $\{a, c\}$. Such scenarios occur frequently in real-world applications, e.g., in model checking (see [130] and references therein) and systems biology [115]. The requirements of such applications, in which output-irrelevant variables have their proper combinatorics, are not properly addressed by approaches exhaustively enumerating entire solutions.[9]

While solution recording as in CDNL-RECORDING can easily be adapted to enumerating solutions distinct on a set P of relevant atoms, simply by replacing the condition

[9]For instance, with the application in [115], the output-irrelevant variables are not functionally dependent from relevant variables, and ambiguities cannot be eliminated by means of symmetry breaking (cf. [205]). As a consequence, plenty solutions may turn out to be indistinguishable.

$var(\sigma_p) \in atom(\Pi)$ in Line 15 of Algorithm 4.5 by $var(\sigma_p) \in P$, it is not straightforward with backtracking-based enumeration methods, in particular, if we do not restrict decision heuristics a priori. The latter is done, e.g., in the "Important First Decision Procedure" [130] and in OPTSAT [125] (which uses a decision heuristic preferring minimal elements of a partially ordered set for computing optimal solutions). Unlike them, we do not make any particular assumptions about the heuristic used and, as required in Section 4.3.1, merely assume the variable in a selected entry to be unassigned and occur in the input.

In the following, we let $\mathbf{A}^P = \{\sigma_p \in \mathbf{A} \mid var(\sigma_p) \in P\}$ denote the *projection* of an assignment $\mathbf{A}$ (or a set $\mathbf{A}$ of entries) to a set P of atoms.[10] The major particularities of enumerating projections are described next.

First, two distinct solutions $\mathbf{A}$ and $\mathbf{B}$ for a set Δ of nogoods can be such that they agree on relevant variables in P and differ only on irrelevant variables outside P (implicating a different decision on some variable outside P):

1. For a solution $\mathbf{A}$ for a set Δ of nogoods such that $\mathbf{A}$ contains $\{\sigma_1, \ldots, \sigma_j\}$ as decision entries, it is possible that $\mathbf{A}^P \subseteq \mathbf{B}$ and $\{\overline{\sigma_1}, \ldots, \overline{\sigma_j}\} \cap \mathbf{B} \neq \emptyset$ hold for some distinct solution $\mathbf{B}$ for Δ. Then, if $\overline{\sigma_i} \in \mathbf{B}$ for $1 \leq i \leq j$, we have that $var(\sigma_i) \notin P$. From this, we conclude that flipping some entry in $\{\sigma_i \mid 1 \leq i \leq j, var(\sigma_i) \notin P\}$ may be insufficient to exclude repetitions of $\mathbf{A}^P$.

Second, two solutions $\mathbf{A}$ and $\mathbf{B}$ for a set Δ of nogoods can be such that they differ on relevant variables in P but agree on decision entries over P (implicating a different decision on some variable outside P):

2. For a solution $\mathbf{A}$ for a set Δ of nogoods such that $\mathbf{A}$ contains $\{\sigma_1, \ldots, \sigma_j\}$ as decision entries, it is possible that $\mathbf{A}^P \not\subseteq \mathbf{B}$ and $\{\sigma_1, \ldots, \sigma_j\}^P \subseteq \mathbf{B}$ hold for some distinct solution $\mathbf{B}$ for Δ. Then, $\mathbf{B}$ includes all decision entries over P from $\mathbf{A}$ but still differs from $\mathbf{A}$ on relevant variables in P. From this, we conclude that flipping some entry in $\{\sigma_1, \ldots, \sigma_j\}^P$ may eliminate a distinguishable and thus non-redundant solution.

Combining the above observations, we note that flipping decision entries over variables outside P may tolerate indistinguishable solutions, while flipping decision entries over P might sacrifice distinguishable ones. Hence, with a decision heuristic selecting freely among unassigned variables, we cannot know which decision entry of a solution ought to be flipped in order to perform enumeration. This obscurity could be avoided by deciding variables in P before those outside P, as proposed in [130]. On unsatisfiable inputs, such approaches however suffer from a negative proof complexity result:

3. Any decision heuristic that selects an entry σ_d such that $var(\sigma_d) \notin P$ only w.r.t. assignments $\mathbf{A}$ such that

$$\bigcup_{\delta \in \Delta, \forall \sigma \in \delta : \overline{\sigma} \notin \mathbf{A}} \{\sigma_p \in \delta \setminus \mathbf{A} \mid var(\sigma_p) \in P\} = \emptyset \tag{4.1}$$

(that is, $var(\sigma) \notin P$ holds for all unassigned entries σ occurring in nogoods δ of Δ not yet excluded by $\mathbf{A}$), on certain inputs, incurs exponentially less efficient best-case computations than an unrestricted decision heuristic.

[10] For an answer set X of a normal program Π, we sometimes call the intersection $X \cap P$ the projection of X to P, given that $X \cap P$ and $(\{\boldsymbol{T}p \mid p \in X\} \cup \{\boldsymbol{F}p \mid p \in atom(\Pi) \setminus X\})^P$ are directly correlated.

This handicap follows from Lemma 3 in [143], showing that CDCL with decisions restricted to variables P acting as input gates of a Boolean circuit has exponentially greater minimum-length proofs of unsatisfiability than DPLL on the infinite family $\{\mathrm{EPHP}_n^{n+1}\}$ of constrained Boolean circuits. A circuit of this family induces a set Δ of nogoods such that every assignment $\mathbf{A}$ satisfying (4.1) yields an immediate conflict by unit propagation. Hence, any restricted decision heuristic is doomed to select only entries σ_d such that $var(\sigma_d) \in P$, and thus it deteriorates search in the sense of Lemma 3 in [143].

The previous observation tells us that performing enumeration of solutions distinct on a set of relevant atoms by altering SELECT in Line 27 of Algorithm 4.6 would drastically degrade the (theoretic) power of search. This motivates us not to apply any such modification inside CDNL-ENUMERATION but to instead devise a dedicated algorithm for the enumeration of answer sets projected to certain relevant atoms.

Our main procedure for enumerating projections of answer sets to a set P of "output" atoms, CDNL-PROJECTION, is shown in Algorithm 4.7. Similar to CDNL-ENUMERATION in Algorithm 4.6, it uses a backtracking level bl to indicate entries that must not be retracted upon backjumping. In fact, the outline of CDNL-PROJECTION partially matches the one of CDNL-ENUMERATION, and the following table displays correspondences between particular cases handled in their algorithms:

No.	Case	Algorithm 4.6	**Algorithm 4.7**
1	Backjumping from conflict	Line 9–12	Line 9–12
2	Backtracking from conflict/solution	Line 14–17 and 22–25	Line **14**–18 and **24**–28
3	Backtracking level introduction	—	Line **30–37**
4	Decision	Line 27–30	Line 39–41

The cases in which both algorithms differ significantly and the corresponding lines of Algorithm 4.7 are highlighted in bold face in the above table. To begin with, we note that Case 1, backjumping from conflict, is handled identically in Algorithm 4.6 and 4.7 by taking the maximum of the assertion level k, returned by CONFLICTANALYSIS in Line 9, and bl as the decision level dl to resume from after backjumping in Line 12. A minor difference in Case 4, (heuristic) decision, is that Algorithm 4.6 retains a selected entry σ_d in $decision(dl)$ (Line 29 of Algorithm 4.6), which is omitted in Algorithm 4.7. Rather than recalling (arbitrary) decision entries for later on flipping them upon backtracking, Algorithm 4.7 includes Case **3** for the introduction of backtracking levels. This case, having no counterpart in Algorithm 4.6, is detailed next.

As mentioned above, the difficulty arising with an a priori unrestricted decision heuristic is that we can, in general, not be sure whether flipping decision entries in a solution $\mathbf{A}$ preserves distinguishable solutions and at the same time excludes repetitions of $\mathbf{A}^P$. The central idea to cope with this is to a posteriori fix an entry $\sigma_p \in \mathbf{A}$ such that $var(\sigma_p) \in P$ in order to perform systematic backtracking on it. In fact, if Line 30–37 of Algorithm 4.7 are executed after printing the true atoms of $\mathbf{A}^P$ in Line 20, then the tests in Line 22 and 23, made beforehand, check that the set $\{\sigma_p \in \mathbf{A}^P \mid bl < dlevel(\sigma_p)\}$ is non-empty. Hence, an entry σ_d from this set can be picked as a (fake) decision in Line 34 to reassign it in Line 37. Given this reassignment, $\mathbf{A}^P$ is in Line 32 *temporarily* recorded in ∇ as $nogood(bl)$, a nogood associated with the new backtracking level bl (obtained by incrementation in Line 30) that denies assignments containing $\mathbf{A}^P$. In fact, after backtracking in Line 33 and establishing σ_d as decision entry at bl in Line 35–37,

Algorithm 4.7: CDNL-PROJECTION

Input : A normal program Π, a set P of atoms, and a number s of projected answer sets to enumerate.

```
1  A := ∅                                                  // ordered assignment over atom(Π) ∪ body(Π)
2  ∇ := ∅                                                  // set of recorded nogoods
3  bl := dl := 0                                           // (systematic) backtracking and decision level
4  loop
5      (A, ∇) := NogoodPropagation(Π, ∇, A)
6      if ε ⊆ A for some ε ∈ Δ_Π ∪ ∇ then                   // conflict
7          if max({dlevel(σ) | σ ∈ ε} ∪ {0}) = 0 then exit
8          if bl < max{dlevel(σ) | σ ∈ ε} then
9              (δ, k) := ConflictAnalysis(ε, Π, ∇, A)
10             ∇ := ∇ ∪ {δ}                                // (temporarily) record conflict nogood
11             dl := max{k, bl}
12             A := A \ {σ ∈ A | dl < dlevel(σ)}           // (bounded) backjumping
13         else
14             ∇ := ∇ \ {nogood(bl)}                       // delete for polynomial space complexity
15             σ_d := decision(bl)
16             dlevel(σ̄_d) := bl := dl := bl − 1
17             A := A \ {σ ∈ A | bl < dlevel(σ)}           // backtracking
18             A := A ∘ σ̄_d                                // flipping
19     else if A^T ∪ A^F = atom(Π) ∪ body(Π) then           // answer set
20         print A^T ∩ P
21         s := s − 1
22         if s = 0 or max({dlevel(σ_p) | σ_p ∈ A^P} ∪ {0}) = 0 then exit
23         if max{dlevel(σ_p) | σ_p ∈ A^P} = bl then
24             ∇ := ∇ \ {nogood(bl)}                       // delete for polynomial space complexity
25             σ_d := decision(bl)
26             dlevel(σ̄_d) := bl := dl := bl − 1
27             A := A \ {σ ∈ A | bl < dlevel(σ)}           // backtracking
28             A := A ∘ σ̄_d                                // flipping
29         else
30             bl := bl + 1
31             nogood(bl) := A^P
32             ∇ := ∇ ∪ {nogood(bl)}                       // (temporarily) record projection
33             A := A \ {σ ∈ A | bl ≤ dlevel(σ)}           // backtracking
34             let σ_d ∈ nogood(bl) \ A in                 // (fake) decision
35                 dlevel(σ_d) := dl := bl
36                 decision(bl) := σ_d
37                 A := A ∘ σ_d
38     else
39         σ_d := Select(Π, ∇, A)                          // decision
40         dlevel(σ_d) := dl := dl + 1
41         A := A ∘ σ_d
```

the presence of $nogood(bl)$ in ∇ excludes a repetition of $\mathbf{A}^P$. On the other hand, since no (decision) entry of the solution $\mathbf{A}$ has yet been flipped, distinguishable solutions not containing $\mathbf{A}^P$ are not eliminated, neither by $nogood(bl)$ nor due to (rash) backtracking.

Finally, we describe Case **2**, backtracking from a conflict/solution, upon which the decision entry at a backtracking level is flipped. Backtracking is triggered either if solely the entries assigned at decision levels up to bl (along with the temporarily recorded nogoods

associated with such levels) cause a conflict, i.e., if the test in Line 8 of Algorithm 4.7 fails, or if the entries in the projection of a solution are all assigned at levels up to bl, as checked in Line 23. In either case, the search space for projections that include all entries assigned at levels up to bl is exhausted, so that some decision entry needs to be flipped. Albeit the flipping of $decision(bl)$ performed in Line 15–18 or 25–28, respectively, is like in CDNL-ENUMERATION (Line 14–17 or 22–25 of Algorithm 4.6), it is noteworthy to mention that $decision(bl)$ here belongs to an explicitly introduced backtracking level and that $var(decision(bl))$ is contained in P. As the same applies to all decision entries at levels smaller than bl, the temporarily recorded nogood associated with bl, $nogood(bl)$, does not exclude any assignment constructible in the sequel. Hence, $nogood(bl)$ can safely be deleted from ∇ without risking the repetition of some already enumerated projection. This deletion on backtracking, done in Line 14 or 24, respectively, distinguishes CDNL-PROJECTION from CDNL-RECORDING in Algorithm 4.5: while the latter is prone to blow up in space, this is not the case with CDNL-PROJECTION because the number of backtracking levels and temporarily recorded nogoods associated with them is tightly bounded by $|P \cap atom(\Pi)|$. Since the number of entries in a temporarily recorded projection is likewise bounded by $|P \cap atom(\Pi)|$, it follows that CDNL-PROJECTION can be run in polynomial space; in view of the considerations in Section 4.3.5, the space complexity of CDNL-PROJECTION is (at most) quadratic in the size of Π.

Example 4.12. *The main steps in a computation of all projections of answer sets of* Π_{11} *from Example 4.10 to* $\{a, b, c\}$ *with* CDNL-PROJECTION$(\Pi_{11}, \{a, b, c\}, 0)$ *are shown in Table 4.8, which indicates macroscopic steps in the same fashion as Table 4.6 and 4.7 for* CDNL-RECORDING$(\Pi_{11}, 0)$ *and* CDNL-ENUMERATION$(\Pi_{11}, 0)$*, respectively. We below mainly focus on describing the differences between the dedicated computation of projected answer sets and algorithms enumerating (non-projected) answer sets; an unabridged trace of* CDNL-PROJECTION$(\Pi_{11}, \{a, b, c\}, 0)$ *is provided in Appendix A.3.*

At the start, the computation of CDNL-PROJECTION *proceeds like* CDNL-RECORDING *and* CDNL-ENUMERATION*, encountering a conflict after picking* $\boldsymbol{T}y$*,* $\boldsymbol{T}a$*, and* $\boldsymbol{T}x$ *as the first three decision entries and, as a consequence of backjumping to decision level* 1 *and picking* $\boldsymbol{T}b$ *as decision entry at level* 2*, obtaining* $\{y, a, b\}$ *as the first answer set of* Π_{11}*. Its projection to* $\{a, b, c\}$*, viz.,* $\{a, b\}$*, is printed in Line 20 of Algorithm 4.7. Furthermore, the tests in Line 22 and 23 fail because* s *is decremented to* -1 *in Line 21 and the output-relevant atom* b *(as well as* c*) is assigned at decision level* 2*. Hence,* CDNL-PROJECTION *does not exit in Line 22 and proceeds in Line 30–32 by temporarily recording* $\{\boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}c\}$ *in* ∇ *as* $nogood(1)$*, associated with the new backtracking level* $bl = 1$*. (The value of* bl *is highlighted by writing* $\mathbf{1}$ *in Table 4.8.) Afterwards, all entries are retracted in Line 33, given that none of them has been assigned at decision level* 0*, and* $decision(1) = \boldsymbol{T}b$ *is picked to be reassigned in Line 37 as decision entry at* $bl = 1$*.*

After picking $\boldsymbol{F}a$ *as decision entry at level* 2*, we encounter a conflict such that* CONFLICTANALYSIS *in Line 9 of Algorithm 4.7 returns* $\{\boldsymbol{F}a\}$ *along with* $k = 0$*. Since the backtracking level* bl *is* 1*, it is in Line 11 taken as the decision level* dl *to resume from after backjumping. That is, the backjump that would in* CDNL-ASP *and* CDNL-RECORDING *lead to assertion level* 0 *is, as with* CDNL-ENUMERATION*, bounded by the backtracking level. However, after inserting* $\boldsymbol{T}a$ *into the assignment at decision level* 0 *(within* NOGOODPROPAGATION*),* $\boldsymbol{T}c$ *becomes unit-resulting for the temporarily recorded nogood* $\{\boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}c\}$*, reflecting that the projected answer set* $\{a, b\}$ *ought*

dl	**A**	δ	Info	Line
1	$\boldsymbol{T}y$			41
	...			5
2	$\boldsymbol{T}a$			41
3	$\boldsymbol{T}x$			41
	...			5
		$\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	6
		$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	dl=1	9
1	$\boldsymbol{T}y$			
	...			
	$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	∇	5
	...			5
	$\boldsymbol{T}a$	$\{\boldsymbol{F}a, \boldsymbol{T}\{not\ x\}\}$	$\Delta_{\Pi_{11}}$	5
2	$\boldsymbol{T}b$			41
	...			5
	$\boldsymbol{F}c$	$\{\boldsymbol{T}\{not\ c\}, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{a, b\}}$	20
		$\{\boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}c\}$	$nogood(\mathbf{1})$	32
1	$\boldsymbol{T}b$		$decision(\mathbf{1})$	37
	...			5
2	$\boldsymbol{F}a$			41
	...			5
		$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	6
		$\{\boldsymbol{F}a\}$	dl=**1**	9
0	$\boldsymbol{T}a$	$\{\boldsymbol{F}a\}$	∇	5
1	$\boldsymbol{T}b$			
	...			
	$\boldsymbol{T}c$	$\{\boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}c\}$	$nogood(\mathbf{1})$	5
	...			5
			$\underline{\{a, b, c\}}$	20
		~~$\{\boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}c\}$~~	$nogood(\mathbf{1})$	24
0	$\boldsymbol{T}a$	$\{\boldsymbol{F}a\}$	∇	
	$\boldsymbol{F}b$			28
	...			5
	$\boldsymbol{T}c$	$\{\boldsymbol{F}\{not\ c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	5
	...			5
1	$\boldsymbol{T}y$			41
	...			5
			$\underline{\{a, c\}}$	20

Table 4.8: Main steps in a computation of all projected answer sets with CDNL-PROJECTION($\Pi_{11}, \{a, b, c\}, 0$).

not be repeated. Under these prerequisites, we obtain $\{x, a, b, c\}$ *as the next answer set of* Π_{11}*. After printing its projection to* $\{a, b, c\}$*, viz.,* $\{a, b, c\}$*, in Line 20, the test in Line 22 fails, while the one in Line 23 succeeds because all (output) atoms are assigned up to backtracking level* 1*. Hence,* $nogood(1) = \{\boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}c\}$ *is deleted from* ∇ *in Line 24, and the complement* $\boldsymbol{F}b$ *of* $decision(1) = \boldsymbol{T}b$ *is assigned in Line 28 at the decremented backtracking level* 0*.*

In the sequel, NOGOODPROPAGATION *derives* $\boldsymbol{T}c$ *at decision level* 0*. A final decision on* $\boldsymbol{T}y$ *leads to the answer set* $\{y, a, c\}$ *of* Π_{11}*. After printing its projection to* $\{a, b, c\}$*, viz.,* $\{a, c\}$*, in Line 20, the test in Line 22 succeeds because all output atoms,* a*,* b*, and* c*, are assigned at decision level* 0*. This shows that flipping* $\boldsymbol{T}y$ *cannot lead to any answer set differing from already enumerated ones on output-relevant atoms. Hence, Algorithm 4.7 finishes in Line 22 without investigating the alternative of flipping* $\boldsymbol{T}y$ *to* $\boldsymbol{F}y$*. Overall,* $\{a, b\}$*,* $\{a, b, c\}$*, and* $\{a, c\}$ *are obtained as the three projections to* $\{a, b, c\}$ *of the five answer sets of* Π_{11}*:* $\{x, a, b, c\}$*,* $\{y, a, b\}$*,* $\{y, a, c\}$*,* $\{z, a, b\}$*, and* $\{z, a, c\}$*.*

In summary, we note that CDNL-PROJECTION combines the principles of the three main algorithms introduced before, CDNL-ASP, CDNL-RECORDING, and CDNL-ENUMERATION. It parallels CDNL-ASP when applied to the decision problem of answer set existence (providing 1 for the third argument s) because it does neither interfere with search nor incur any computational overhead a priori. In particular, the fact that decision heuristics (SELECT in Line 39 of Algorithm 4.7) remain uninfluenced keeps CDNL-PROJECTION out of the realm of a negative proof complexity result in [143]. With CDNL-RECORDING, it has the usage of solution-suppressing nogoods in common. But since such nogoods are only temporarily recorded, CDNL-PROJECTION can still be run in polynomial space. Space requirements are limited by, similar to CDNL-ENUMERATION, performing systematic backtracking in order to keep track of enumerated as well as yet uninvestigated (projections of) solutions. As shown in the next section, CDNL-PROJECTION is complete and does not repeat projected answer sets, provided that a nogood associated with a backtracking level is not deleted before the level is retracted. Since the latter is easy to grant, we have achieved the design goals for enumeration algorithms postulated at the beginning of Section 4.4. To our knowledge, CDNL-PROJECTION is the first method to enumerate projections of solutions that jointly complies with all of these design goals.

4.4.4 Soundness and Completeness

In the following, we investigate properties of the introduced enumeration algorithms. Since their intentions are partially different, enumerating entire answer sets versus projections of them, the following concepts of *soundness*, *completeness*, and *redundancy-freeness* take the "visibility" of atoms (also considered, e.g., in [58, 136]) into account.

Definition 4.1. *Let* Π *be a normal program and* P *a set of atoms.*
Then, we define an enumeration algorithm as

1. *sound w.r.t.* P *if, for every printed set* Y *of atoms, some answer set* X *of* Π *satisfies* $X \cap P = Y$*;*

2. *complete w.r.t.* P *if, for every answer set* X *of* Π*, some printed set* Y *of atoms satisfies* $Y = X \cap P$*;*

3. *redundancy-free w.r.t. P if every pair Y_1, Y_2 of sets of atoms printed one after the other satisfies $Y_1 \cap P \neq Y_2 \cap P$.*

Starting with the simplest approach to answer set enumeration, CDNL-RECORDING, its invariants resemble those of CDNL-ASP stated in Lemma 4.11 on Page 75.

Lemma 4.13. *Let Π be a normal program and s an integer.*

Then, we have that the following holds whenever Line 5 of Algorithm 4.5 is entered in an execution of CDNL-RECORDING(Π, s)*:*

1. *∇ is a set of nogoods such that $\delta \not\subseteq \mathbf{B}$ for every $\delta \in \nabla$ and any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi)$ has not yet been printed;*

2. *$\mathbf{A}$ is an ordered assignment such that all decision levels are implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$.*

In view of the fact that enumerated solutions are suppressed by persistently recorded nogoods, the first item applies only to solutions that have not yet been enumerated (given that their true atoms have not yet been printed). Also note that the number of solutions (i.e., answer sets) may be exponential in the size of Π, so that ∇ can blow up significantly before the residual problem turns out to be unsatisfiable, meaning that there is no unenumerated solution left.

From Lemma 4.13, we derive the following main result for CDNL-RECORDING.

Theorem 4.14. *Let Π be a normal program.*

Then, we have that CDNL-RECORDING$(\Pi, 0)$ *is terminating as well as sound, complete, and redundancy-free w.r.t. $atom(\Pi)$.*

Unlike CDNL-RECORDING (presented only for comparison), our primary algorithm for enumerating entire answer sets, CDNL-ENUMERATION, applies a dedicated backtracking scheme and can thus be run in polynomial space. Its important invariants can be summarized as follows.

Lemma 4.15. *Let Π be a normal program and s an integer.*

Then, we have that the following holds whenever Line 5 of Algorithm 4.6 is entered in an execution of CDNL-ENUMERATION(Π, s)*:*

1. *∇ is a set of nogoods such that $\delta \not\subseteq \mathbf{B}$ for every $\delta \in \nabla$ and any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$;*

2. *$\mathbf{A}$ is an ordered assignment such that all decision levels greater than bl are implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$;*

3. *if $\mathbf{A} \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi)$ has not yet been printed, then $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\}$ and $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\}) \notin \mathbf{B}$.*

Note that the first item is similar to Lemma 4.11 applying to CDNL-ASP: none of the dynamic nogoods in ∇ ever eliminates an answer set of Π, no matter whether a corresponding solution has already been enumerated. In fact, the repetition of an enumerated solution is suppressed by successively flipping its decision entries, where complements are assigned at the backtracking level bl. Such deliberate flips introduce non-decision entries lacking antecedents, so that only the decision levels beyond bl are implied, as stated

in the second item. Finally, the third item tells us that any first entry σ_d on which the current assignment $\mathbf{A}$ differs from a not yet enumerated solution $\mathbf{B}$ must be a decision entry, so that enumerating $\mathbf{B}$ is made possible when flipping σ_d to $\overline{\sigma_d}$.

From Lemma 4.15, we derive the following main result for CDNL-ENUMERATION.

Theorem 4.16. *Let Π be a normal program.*

Then, we have that CDNL-ENUMERATION$(\Pi, 0)$ *is terminating as well as sound, complete, and redundancy-free w.r.t. $atom(\Pi)$.*

Regarding CDNL-PROJECTION, aiming at the enumeration of projected answer sets, important invariants can be summarized as follows.

Lemma 4.17. *Let Π be a normal program, P a set of atoms, and s an integer.*

Then, we have that the following holds whenever Line 5 of Algorithm 4.7 is entered in an execution of CDNL-PROJECTION(Π, P, s)*:*

1. *∇ is a set of nogoods such that $\delta \not\subseteq \mathbf{B}$ for every $\delta \in \nabla$ and any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed;*

2. *$\mathbf{A}$ is an ordered assignment such that all decision levels greater than bl are implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$;*

3. *if $\mathbf{A} \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed, then $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\}$;*

4. *if $\mathbf{A} \not\subseteq \mathbf{B}$ and $\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\} \leq bl$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed, then $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\}) \in (\mathbf{A} \setminus \mathbf{B})^P$.*

The invariants in Lemma 4.17 can be viewed as a combination of those stated in Lemma 4.13 and 4.15. Similar to Lemma 4.13, the first item applies only to solutions such that their projections to P have not yet been enumerated. However, the solution-suppressing nogoods utilized by CDNL-PROJECTION are only temporarily recorded and deleted upon backtracking. Since backtracking is accompanied by deliberately flipping some decision entry, like in Lemma 4.15, the second item applies only to decision levels beyond bl. Still paralleling Lemma 4.15, any first entry σ_d on which the current assignment $\mathbf{A}$ differs from a solution $\mathbf{B}$ whose projection to P has not yet been enumerated must be assigned at a decision level greater than 0. In addition, if $dlevel(\sigma_d) \leq bl$, the fourth item tells us that σ_d is a decision entry such that $var(\sigma_d) \in P$. As with CDNL-ENUMERATION, this makes sure that enumerating the projection of $\mathbf{B}$ to P is made possible when flipping σ_d to $\overline{\sigma_d}$.

From Lemma 4.17, we derive the following main result for CDNL-PROJECTION.

Theorem 4.18. *Let Π be a normal program and P a set of atoms.*

Then, we have that CDNL-PROJECTION$(\Pi, P, 0)$ *is terminating as well as sound, complete, and redundancy-free w.r.t. P.*

We have thus established the desired formal properties, soundness, completeness, and redundancy-freeness (as well as termination), for the three enumeration algorithms devised in this section. Notably, the enumeration algorithms are not intrusive a priori: before the first answer set is found, they run similar to CDNL-ASP, the decision algorithm

they are based on. To the best of our knowledge, CDNL-ENUMERATION and CDNL-PROJECTION are the first enumeration algorithms combining conflict-driven learning and backjumping according to the First-UIP scheme with dedicated backtracking to meet all of the design goals postulated at the beginning of Section 4.4.

4.5 Experimental Results

Our approach to conflict-driven ASP solving is implemented in *clasp* [95, 97, 99], an award-winning[11] state-of-the-art Boolean constraint solver with ASP as its core area. The *clasp* system integrates and extends the fundamental algorithms presented in Section 4.3 and 4.4. To mention only the most salient features, *clasp* offers advanced preprocessing techniques [96], reasoning modes like optimization and computation of brave or cautious consequences [97], lookback-based decision heuristics like VSIDS [185], BerkMin [127], and VMTF [203], various restart policies [133, 20, 204], progress (or phase) saving [195], lazy data structures [185, 56, 203, 20], failed-literal detection [76, 209], and native treatment of extended rules [209, 72, 84] (including choice constructs in heads of rules as well as cardinality and weight constraints in rule bodies). The efficiency, robustness, and versatility of *clasp* are outstanding in ASP solving and, most likely, the area of Boolean constraint solving in general. The *clasp* system constitutes a central component of *Potassco* [81], the Potsdam Answer Set Solving Collection bundling tools for ASP developed at the University of Potsdam; it is implemented (primarily) by Benjamin Kaufmann and freely available as an open source package at [198].[12]

The experimental results in Section 4.5.1, 4.5.2, and 4.5.3 are borrowed from [99], [94], and [98], respectively, coauthored by the author of this thesis, and are included here to illustrate the effectiveness of the algorithms presented in the previous sections.[13]

4.5.1 Experiments on Decision Algorithm

Our first series of experiments demonstrates the aptitude of CDNL-ASP in Algorithm 4.1 on Page 62 for deciding answer set existence on problems in the "NP Decision" category of the 2009 ASP competition [47]. Our comparison considers *clasp* (version 1.3.1) in its default setting as well as a setting better suited for the benchmarks in focus. The latter, denoted by *clasp*$^+$, invokes *clasp* with the options `--sat-prepro` and `--trans-ext=dynamic`, using SatELite-like preprocessing [54] on nogoods as well as a context-dependent handling of extended rules, excluding "small" extended rules from native treatment (cf. [84]) and rather transforming them into normal rules.

For comparison, we also consider *cmodels* (version 3.79) with *minisat* (version 2.0), *smodels* (version 2.34 with the option `-restart`), as well as *lp2sat* (version 1.13 plus further preprocessing tools[14]) with *minisat* (version 2.0) or *clasp* (version 1.3.1) as un-

[11] Amongst others, *clasp* successfully participated in the 2009 ASP competition [47] (`http://dtai.cs.kuleuven.be/events/ASP-competition`), the 2009 Pseudo-Boolean competition (`http://www.cril.univ-artois.fr/PB09`), and the 2009 SAT competition (`http://www.satcompetition.org`).

[12] The author is deeply grateful to Benjamin Kaufmann and all other contributors for doing excellent and invaluable work in developing and constantly improving *clasp*.

[13] The experiments were conducted by Benjamin Kaufmann [99, 94, 98] and André Neumann [94]. Benchmarks as well as extended results are available at [38].

[14] See `http://dtai.cs.kuleuven.be/events/ASP-competition/Teams/LP2SATMINISAT.shtml` for details.

derlying SAT solver. For *cmodels* and *lp2sat*, we below indicate the use of either *minisat* or *clasp* as underlying SAT solver by adding "[m]" or "[c]," respectively, and the activation of restarts with *smodels* is indicated by writing $smodels_r$. The experiments were run under Linux on an Intel Quad-Core Xeon E5520 machine equipped with 2.27GHz processors. Every benchmark instance was run three times per solver, each run restricted to 600 seconds time and 2GB RAM. A run finished when the solver found an answer set, reported unsatisfiability (no answer set), or exceeded the time or memory limit.

The SAT-based solver *cmodels* [123] converts the completion of a logic program into propositional clauses and delegates the search for a model to a SAT solver. Except for the treatment of extended rules (see below), on tight programs, this approach is comparable to the one of *clasp*. In the non-tight case, *cmodels* delays (sophisticated) unfounded set checks until an assignment is total, while *clasp* and *smodels* integrate them into their propagation. In fact, *smodels* [209] is a traditional ASP solver combining DPLL-style search with an unfounded set checking routine identifying greatest unfounded sets [216]. Finally, *lp2sat* [139] like *cmodels* converts a logic programs into propositional clauses and delegates the search for a model to some SAT solver. On tight programs, *lp2sat*'s translation amounts to completion, while level mappings [136, 187] are used to capture acyclic derivability of atoms from non-tight programs.

Our experimental results are summarized in Table 4.9–4.11, giving average runtimes in seconds and numbers of timed-out runs (in parentheses) for every solver on each benchmark class, with timeouts taken as 600 seconds. While Table 4.9 considers all benchmarks, divided into tight and non-tight ones, Table 4.10 and 4.11 analogously report results restricted to satisfiable and unsatisfiable instances, respectively. Each table gives the number of instances per benchmark class in the column headed by "#." In addition, Table 4.9 provides the respective partition into satisfiable and unsatisfiable instances in parentheses. The last column amounts to the virtual best solver, composed of the smallest average runtime and the smallest number of timeouts observed on each benchmark class. The rows marked with "$\varnothing(\varnothing)$" average runtimes and timeouts over a collection of benchmark classes under consideration.[15] The following row gives the Euclidean distance (in an n-dimensional space, where n is the number of benchmark classes and a point is a column of n average runtimes) of every solver to the virtual best one on the respective collection in focus; the quadratic distance calculation scheme punishes imbalanced and rewards consistent performance more than averaging. Some benchmark classes make heavy use of extended rules, so that their different treatments, e.g., in $clasp^+$ and *cmodels*, have a significant impact on the observed performances; such benchmark classes are marked with "*" in Table 4.9–4.11.

Considering the results on tight benchmarks in the upper part of Table 4.9, we note that traditional ASP solver $smodels_r$ is consistently outperformed by systems exploiting conflict-driven learning. For instance, $smodels_r$ times out on all (satisfiable) instances of *15Puzzle*, which are rather unproblematic for the other solvers. In fact, occasional varying performances of the latter on tight programs are due to different treatments of extended rules and/or determinizations of inherent non-determinisms in *minisat* and *clasp*, respectively. Any such differences may turn out to be advantageous the one or the other way around; e.g., $clasp^+$ and *lp2sat*[c] have an edge on other solvers on *GraphColouring*, *clasp* in its default setting is fastest on *SchurNumbers*, while *lp2sat*[m] yields the fewest timeouts on *WeightBoundedDomSet*.

[15] We provide averages (rather than sums) for balancing diverse numbers of instances per benchmark class.

Benchmark	#	*clasp*	*clasp*$^+$	*cmodels*[m]	*smodels*$_{cr}$	*lp2sat*[m]	*lp2sat*[c]	*virtual best*
15Puzzle	16 (16/0)	33.01 (0)	20.18 (0)	31.36 (0)	600.00 (48)	22.21 (0)	15.13 (0)	15.13 (0)
BlockedNQueens	29 (15/14)	5.09 (0)	4.91 (0)	9.04 (0)	29.37 (0)	13.19 (0)	5.22 (0)	4.91 (0)
ChannelRouting	10 (6/4)	120.13 (6)	120.14 (6)	120.58 (6)	120.90 (6)	121.34 (6)	121.08 (6)	120.13 (6)
EdgeMatching	29 (29/0)	0.23 (0)	0.41 (0)	59.32 (0)	60.32 (0)	13.05 (0)	5.58 (0)	0.23 (0)
Fastfood*	29 (10/19)	1.17 (0)	0.90 (0)	29.22 (0)	83.93 (3)	46.85 (0)	24.95 (0)	0.90 (0)
GraphColouring	29 (9/20)	421.55 (60)	357.88 (39)	422.66 (57)	453.77 (63)	409.70 (51)	357.57 (39)	357.57 (39)
Hanoi	15 (15/0)	11.76 (0)	3.97 (0)	2.92 (0)	523.77 (39)	3.81 (0)	5.36 (0)	2.92 (0)
HierarchicalClustering*	12 (8/4)	0.16 (0)	0.17 (0)	0.76 (0)	1.56 (0)	0.94 (0)	0.86 (0)	0.16 (0)
SchurNumbers	29 (13/16)	17.44 (0)	49.60 (0)	75.70 (0)	504.17 (72)	90.88 (6)	36.93 (0)	17.44 (0)
Solitaire	27 (22/5)	204.78 (27)	162.82 (21)	175.69 (21)	316.96 (36)	222.60 (27)	210.14 (27)	162.82 (21)
Sudoku	10 (10/0)	0.15 (0)	0.16 (0)	2.55 (0)	0.25 (0)	0.87 (0)	0.82 (0)	0.15 (0)
WeightBoundedDomSet*	29 (29/0)	123.13 (15)	102.18 (12)	300.26 (36)	400.84 (51)	179.56 (9)	143.87 (12)	102.18 (9)
∅ (∅) *(tight)*	264 (182/82)	78.22 (9.00)	68.61 (6.50)	102.50 (10.00)	257.99 (26.50)	93.75 (8.25)	77.29 (7.00)	65.38 (6.25)
Eucl. dist. (tight)		81.80	32.58	227.19	991.76	141.67	70.51	0.00
ConnectedDomSet*	21 (10/11)	40.42 (3)	36.11 (3)	7.46 (0)	183.76 (15)	13.43 (0)	13.62 (0)	7.46 (0)
GeneralizedSlitherlink*	29 (29/0)	0.10 (0)	0.22 (0)	1.92 (0)	0.16 (0)	5.05 (0)	12.90 (0)	0.10 (0)
GraphPartitioning*	13 (6/7)	9.27 (0)	7.98 (0)	20.19 (0)	92.10 (3)	365.18 (21)	344.39 (21)	7.98 (0)
HamiltonianPath	29 (29/0)	0.07 (0)	0.06 (0)	0.21 (0)	2.22 (0)	3.45 (0)	15.68 (0)	0.06 (0)
KnightTour	10 (10/0)	124.29 (6)	91.80 (3)	242.48 (12)	150.55 (3)	545.42 (27)	487.61 (24)	91.80 (3)
Labyrinth	29 (29/0)	123.82 (12)	82.92 (6)	142.24 (6)	594.10 (81)	282.23 (27)	534.62 (75)	82.92 (6)
MazeGeneration	29 (10/19)	91.17 (12)	89.89 (12)	90.41 (12)	293.62 (42)	125.94 (9)	85.57 (6)	85.57 (6)
Sokoban	29 (9/20)	0.73 (0)	0.80 (0)	3.39 (0)	176.01 (15)	6.11 (0)	3.99 (0)	0.73 (0)
TravellingSalesperson*	29 (29/0)	0.05 (0)	0.06 (0)	317.82 (7)	0.22 (0)	441.68 (55)	198.34 (9)	0.05 (0)
WireRouting	23 (12/11)	42.81 (3)	36.36 (3)	175.73 (12)	448.32 (45)	460.89 (48)	459.97 (51)	36.36 (3)
∅ (∅) *(non-tight)*	241 (173/68)	43.27 (3.60)	34.62 (2.70)	100.19 (4.90)	194.11 (20.40)	224.94 (18.70)	215.67 (18.60)	31.30 (1.80)
Eucl. dist. (non-tight)		62.37	28.97	383.16	739.35	866.08	832.52	0.00
∅ (∅)	505 (355/150)	62.33 (6.55)	53.16 (4.77)	101.45 (7.68)	228.95 (23.73)	153.38 (13.00)	140.19 (12.27)	49.89 (4.23)
Eucl. dist.		102.86	43.59	445.45	1237.02	877.59	835.50	0.00

Table 4.9: Average runtimes on benchmarks of the 2009 ASP competition.

Benchmark	#	*clasp*	*clasp*$^+$	*cmodels*[m]	*smodels*$_r$	*lp2sat*[m]	*lp2sat*[c]	*virtual best*
15Puzzle	16	33.01 (0)	20.18 (0)	31.36 (0)	600.00 (48)	22.21 (0)	15.13 (0)	15.13 (0)
BlockedNQueens	15	3.48 (0)	4.93 (0)	7.52 (0)	22.13 (0)	13.99 (0)	4.16 (0)	3.48 (0)
ChannelRouting	6	0.16 (0)	0.17 (0)	0.67 (0)	1.35 (0)	1.63 (0)	1.37 (0)	0.16 (0)
EdgeMatching	29	0.23 (0)	0.41 (0)	59.32 (0)	60.32 (0)	13.05 (0)	5.58 (0)	0.23 (0)
Fastfood*	10	0.12 (0)	0.49 (0)	9.26 (0)	82.44 (0)	45.33 (0)	18.54 (0)	0.12 (0)
GraphColouring	9	24.99 (0)	32.66 (0)	98.64 (3)	128.80 (3)	57.93 (0)	31.27 (0)	24.99 (0)
Hanoi	15	11.76 (0)	3.97 (0)	2.92 (0)	523.77 (39)	3.81 (0)	5.36 (0)	2.92 (0)
HierarchicalClustering*	8	0.14 (0)	0.13 (0)	1.02 (0)	1.52 (0)	1.20 (0)	1.03 (0)	0.13 (0)
SchurNumbers	13	37.25 (0)	109.13 (0)	166.77 (0)	600.00 (39)	200.49 (6)	80.40 (0)	37.25 (0)
Solitaire	22	114.96 (12)	63.46 (6)	79.25 (6)	252.63 (21)	136.82 (12)	121.54 (12)	63.46 (6)
Sudoku	10	0.15 (0)	0.16 (0)	2.55 (0)	0.25 (0)	0.87 (0)	0.82 (0)	0.15 (0)
WeightBoundedDomSet*	29	123.13 (15)	102.18 (12)	300.26 (36)	400.84 (51)	179.56 (9)	143.87 (12)	102.18 (9)
∅ (∅) *(tight, sat)*	182	29.11 (2.25)	28.16 (1.50)	63.29 (3.75)	222.84 (16.75)	56.41 (2.25)	35.76 (2.00)	20.85 (1.25)
Eucl. dist. (tight, sat)		59.07	72.48	256.02	1037.56	203.65	85.96	0.00
ConnectedDomSet*	10	9.28 (0)	1.74 (0)	12.06 (0)	135.10 (6)	17.57 (0)	12.36 (0)	1.74 (0)
GeneralizedSlitherlink*	29	0.10 (0)	0.22 (0)	1.92 (0)	0.16 (0)	5.05 (0)	12.90 (0)	0.10 (0)
GraphPartitioning*	6	0.11 (0)	0.14 (0)	4.52 (0)	0.56 (0)	114.21 (3)	118.25 (3)	0.11 (0)
HamiltonianPath	29	0.07 (0)	0.06 (0)	0.21 (0)	2.22 (0)	3.45 (0)	15.68 (0)	0.06 (0)
KnightTour	10	124.29 (6)	91.80 (3)	242.48 (12)	150.55 (3)	545.42 (27)	487.61 (24)	91.80 (3)
Labyrinth	29	123.82 (12)	82.92 (6)	142.24 (6)	594.10 (81)	282.23 (27)	534.62 (75)	82.92 (6)
MazeGeneration	10	0.07 (0)	0.08 (0)	0.08 (0)	0.15 (0)	114.32 (0)	10.92 (0)	0.07 (0)
Sokoban	9	0.63 (0)	0.78 (0)	5.40 (0)	320.54 (9)	10.77 (0)	4.34 (0)	0.63 (0)
TravellingSalesperson*	29	0.05 (0)	0.06 (0)	317.82 (7)	0.22 (0)	441.68 (55)	198.34 (9)	0.05 (0)
WireRouting	12	74.00 (3)	62.63 (3)	134.94 (3)	407.44 (18)	513.20 (30)	519.94 (30)	62.63 (3)
∅ (∅) *(non-tight, sat)*	173	33.24 (2.10)	24.04 (1.20)	86.17 (2.80)	161.10 (11.70)	204.79 (14.20)	191.49 (14.10)	24.01 (1.20)
Eucl. dist. (non-tight, sat)		53.99	0.20	364.12	709.78	818.54	789.78	0.00
∅ (∅) *(sat)*	355	30.99 (2.18)	26.29 (1.36)	73.69 (3.32)	194.78 (14.45)	123.85 (7.68)	106.55 (7.50)	22.29 (1.23)
Eucl. dist. (sat)		80.02	72.48	445.12	1257.11	843.49	794.44	0.00

Table 4.10: Average runtimes on *satisfiable* benchmarks of the 2009 ASP competition.

Benchmark	#	*clasp*		*clasp*$^+$		*cmodels*[m]		*smodels*$_r$		*lp2sat*[m]		*lp2sat*[c]		*virtual best*	
BlockedNQueens	14	6.82	(0)	4.88	(0)	10.68	(0)	37.14	(0)	12.33	(0)	6.35	(0)	4.88	(0)
ChannelRouting	4	300.10	(6)	300.10	(6)	300.44	(6)	300.23	(6)	300.90	(6)	300.66	(6)	300.10	(6)
*Fastfood**	19	1.72	(0)	1.11	(0)	39.72	(0)	84.71	(3)	47.66	(0)	28.32	(0)	1.11	(0)
GraphColouring	20	600.00	(60)	504.24	(39)	568.48	(54)	600.00	(60)	567.99	(51)	504.40	(39)	504.24	(39)
*HierarchicalClustering**	4	0.21	(0)	0.24	(0)	0.24	(0)	1.63	(0)	0.42	(0)	0.50	(0)	0.21	(0)
SchurNumbers	16	1.34	(0)	1.24	(0)	1.70	(0)	426.30	(33)	1.82	(0)	1.61	(0)	1.24	(0)
Solitaire	5	600.00	(15)	600.00	(15)	600.00	(15)	600.00	(15)	600.00	(15)	600.00	(15)	600.00	(15)
∅ (∅)	82	215.74	(11.57)	201.69	(8.57)	217.32	(10.71)	292.86	(16.71)	218.73	(10.29)	205.98	(8.57)	201.68	(8.57)
Eucl. dist.		95.79		0.03		75.18		444.84		79.30		27.26		0.00	
∅ (∅) *(tight, unsat)*	11	68.73	(3)	67.35	(3)	3.28	(0)	227.99	(9)	9.67	(0)	14.77	(0)	3.28	(0)
*ConnectedDomSet**	7	17.12	(0)	14.70	(0)	33.62	(0)	170.56	(3)	580.29	(18)	538.22	(18)	14.70	(0)
*GraphPartitioning**	19	139.12	(12)	137.16	(12)	137.95	(12)	448.09	(42)	132.06	(9)	124.86	(6)	124.86	(6)
MazeGeneration	20	0.78	(0)	0.81	(0)	2.49	(0)	110.97	(6)	4.01	(0)	3.83	(0)	0.78	(0)
Sokoban	11	8.78	(0)	7.70	(0)	220.23	(9)	492.92	(27)	403.83	(18)	394.56	(21)	7.70	(0)
∅ (∅) *(non-tight, unsat)*	68	46.91	(3.00)	45.54	(3.00)	79.51	(4.20)	290.10	(17.40)	225.97	(9.00)	215.25	(9.00)	30.27	(1.20)
Eucl. dist. (non-tight, unsat)		67.03		65.24		213.78		653.33		690.59		651.05		0.00	
∅ (∅) *(unsat)*	150	145.39	(8.00)	136.63	(6.25)	159.90	(8.00)	291.71	(17.00)	221.75	(9.75)	209.84	(8.75)	130.26	(5.50)
Eucl. dist. (unsat)		116.91		65.24		226.62		790.39		695.13		651.62		0.00	

Table 4.11: Average runtimes on *unsatisfiable* benchmarks of the 2009 ASP competition.

On non-tight benchmarks, we observe that the problem representation overhead incurred by *lp2sat*'s translational approach is a major handicap, though *minisat* and *clasp* may react more or less sensitively (cf. *Labyrinth* and *TravellingSalesperson*). Except for *MazeGeneration*, the strategy of *cmodels* to verify candidate models found by *minisat* already improves on eager translation by *lp2sat*. However, integrating unfounded set checking into propagation is usually even more effective, as it can be observed when comparing *clasp* and *clasp*$^+$ to *cmodels* on *KnightTour* and *WireRouting*. Beyond (non-)tightness, the treatment of extended rules is a crucial factor on some of the considered benchmark classes. Their transformation into normal rules, as done by *cmodels* and *lp2sat*, turns out to be helpful on *ConnectedDomSet*, while it drastically blows up problem representations and thus deteriorates performance on *TravellingSalesperson*.

Focusing on either satisfiable or unsatisfiable instances in Table 4.10 and 4.11, respectively, sheds some light on the distribution of hardness within benchmark classes, yet without exhibiting any overwhelming impact regarding relative solver performances. In fact, a look into Table 4.10, especially at *cmodels* and *smodels*$_r$, reveals that the satisfiable instances of *ChannelRouting*, *GraphPartitioning*, and *MazeGeneration* are rather easy. Interestingly, both *lp2sat* variants still have difficulties with the non-tight benchmarks, viz., *GraphPartitioning* and *MazeGeneration*, indicating that an eager translation of logic programs may diminish search performance. On the other hand, all solvers perform worse on the satisfiable instances of *SchurNumbers* than on the unsatisfiable ones, and the same also applies to *clasp*, *clasp*$^+$, and *lp2sat* on *WireRouting*. Looking at these two classes in Table 4.11, we observe that the unsatisfiable instances of *SchurNumbers* are trivial for all solvers but *smodels*$_r$, while only *clasp* and *clasp*$^+$ complete all of the unsatisfiable *WireRouting* instances. The latter suggests that such instances are not inherently hard but that lacking either conflict-driven learning (*smodels*$_r$) or native unfounded set checking (*cmodels* and *lp2sat*) renders them more difficult.

Unlike with *SchurNumbers* and *WireRouting*, some of the unsatisfiable instances of *ChannelRouting*, *GraphColouring*, *Solitaire*, and *MazeGeneration* turn out to be much harder than their satisfiable counterparts (at least for the considered solvers). Notably, *lp2sat*[c], running *clasp* as SAT solver, completes more unsatisfiable *MazeGeneration* instances than *clasp* and *clasp*$^+$ themselves, which contrasts with the behavior observed on satisfiable instances (cf. *MazeGeneration* in Table 4.10 and 4.11). A similar shift of behaviors is due to one unsatisfiable instance of *ConnectedDomSet*, which poses a problem to the native treatment of extended rules in *clasp* and *clasp*$^+$, while *cmodels* and *lp2sat* do not have any such difficulties. In view of this, we think that the dynamic selection among possible handlings of extended rules (native treatment and/or transformation) is interesting future work.

4.5.2 Experiments on Enumeration Algorithms

Our second series of experiments compares the performances of enumeration algorithms on logic programs having many answer sets. To this end, we consider *clasp* (version RC4) in two different modes, running either CDNL-ENUMERATION in Algorithm 4.6 on Page 81 (denoted by *clasp*^{a}) or CDNL-RECORDING in Algorithm 4.5 on Page 78 (denoted by *clasp*^{b}). Note that *clasp*^{b} implements a slightly optimized form of solution recording: instead of recording the set of all entries over atoms as a nogood, it includes only the decision entries used for constructing a solution. The same strategy is pursued by *smodels*$_{cc}$ [218], but with decisions restricted to atoms. Unlike this, *cmodels* pro-

No.	Instance	#Sol	*clasp*^{a}	*clasp*^a_r	*clasp*^{b}	*clasp*^b_r	*smodels*	*smodels*$_r$	*smodels*$_{cc}$	*cmodels*
1	hc_19	10^4	7.2	7.2	7.7	7.2	•	•	•	•
2	hc_19	10^5	71.4	77.1	83.5	91.2	•	•	•	•
3	hc_20	10^4	9.3	9.5	10.9	9.5	•	•	•	•
4	hc_20	10^5	103.4	117.2	115.8	109.9	•	•	•	•
5	mutex3IDFD	10^5	1.4	1.4	35.4	35.9	5.5	5.8	240.6	•
6	mutex3IDFD	10^6	14.0	13.9	•	•	55.9	52.8	•	•
7	mutex4IDFD	10^4	20.8	27.4	43.8	37.0	44.7	574.7	47.5	•
8	mutex4IDFD	10^5	52.2	63.2	596.7	585.7	273.4	•	•	•
9	pigeon_15	10^5	2.7	2.7	4.0	3.9	7.1	8.6	126.7	•
10	pigeon_15	10^6	26.1	26.5	53.0	54.7	71.8	73.6	•	•
11	pigeon_15	10^7	260.7	262.8	•	•	•	•	•	•
12	pigeon_16	10^5	3.2	3.1	4.4	4.6	7.8	9.9	175.2	•
13	pigeon_16	10^6	30.1	30.5	57.7	59.6	78.5	80.9	•	•
14	pigeon_16	10^7	303.0	304.5	•	•	•	•	•	•
15	queens_19	10^4	14.4	17.1	13.1	15.1	47.1	115.0	49.0	427.5
16	queens_19	10^5	141.5	143.8	135.9	162.7	265.1	358.1	•	•
17	queens_20	10^4	14.1	15.8	13.1	15.3	127.0	172.1	48.3	569.2
18	queens_20	10^5	147.2	170.5	149.6	178.6	380.3	•	•	•
19	schur-n29-m44	10^4	22.4	26.4	19.8	22.7	17.4	49.4	15.6	•
20	schur-n29-m44	10^5	203.1	212.5	177.2	246.4	132.4	175.9	353.2	•
21	schur-n29-m45	10^4	24.7	21.8	21.5	24.6	17.2	50.2	16.1	•
22	schur-n29-m45	10^5	231.6	265.6	190.7	199.9	133.3	176.0	397.3	•

Table 4.12: Experiments enumerating answer sets.

vides whole answer sets as blocking clauses to an underlying (conflict-driven learning) SAT solver [123]. While restarts are disabled in *clasp*^{a} and *clasp*^{b}, we also consider both variants augmented with bounded and unbounded restarts (indicated by an additional subscript "$_r$"), respectively. The bounded restart variant, *clasp*^a_r, is allowed to resume search from the backtracking level (cf. Algorithm 4.6), while *clasp*^b_r can perform unrestricted restarts even after finding an answer set.

For comparison, our experiments additionally incorporate *smodels* (version 2.32) and the variant *smodels*$_r$ with activated `-restart` option, *smodels*$_{cc}$ (version 1.08) with the option `-nolookahead`, as recommended by the developers,[16] and *cmodels* (version 3.67) with *zchaff* (version 2004.11.15). The experiments were run on a 2.2GHz PC under Linux. We report runtimes in seconds, taking the average over ten runs per solver and benchmark instance, each run restricted to 600 seconds time and 512MB RAM.

Table 4.12 displays runtimes taken to enumerate answer sets. The instances stem from the areas of Hamiltonian cycles in complete graphs (No. 1–4), bounded model checking (No. 5–8), pigeon-hole (No. 9–14), n-queens (No. 15–18), and Schur numbers (No. 19–22). We have chosen these combinatorial benchmarks because they admit many answer sets. This allows us to observe the performances of enumeration approaches w.r.t. increasing numbers of answer sets. The number of requested (and successfully enumerated by some solvers) answer sets per run is given in the third column. Timeouts (in all ten runs of a solver) are indicated by "•."

Comparing the variants of *clasp*, we observe that *clasp*^{a} and *clasp*^a_r scale better than *clasp*^{b} and *clasp*^b_r. This is most intelligible on the benchmarks from bounded model checking (No. 5–8) and the pigeon-hole instances (No. 9–14). Solutions for the former contain many decision entries, so that the large nogoods recorded by *clasp*^{b} and *clasp*^b_r

[16] See `http://www.nku.edu/~wardj1/Research/smodels_cc.html`.

slow them down. The pigeon-hole instances are structurally simple, so that all decisions yield solutions; since the amount of easy-to-compute solutions is enormous, the sheer number of recorded nogoods overwhelms *clasp*^{b} and *clasp*^b_r. Also note that *smodels*' efforts on failed-literal detection are unprofitable on the pigeon-hole instances.

On Hamiltonian cycles (No. 1–4), n-queens (No. 15–18), and Schur numbers (No. 19–22), the picture is rather indifferent. That is, the time spent on search tends to dominate enumeration time, and recorded solution-suppressing nogoods are not as critical as with the aforementioned benchmarks. Notably, *smodels* is very effective on Schur numbers. We verified that all *clasp* variants make the same number of decisions as *smodels*; hence, we conjecture that runtime gaps are due to implementation differences (e.g., counter-based propagation in *smodels* versus watched literals in *clasp*). Regarding the other systems, we observe that *smodels*$_{cc}$ usually enumerates slower than *smodels*, but it is sometimes faster when search plays a role (No. 17); the enumeration approach of *cmodels*, recording answer sets, is clearly outperformed. Finally, we note that different restart policies of *clasp*, i.e., *clasp*^{a} versus *clasp*^a_r and *clasp*^{b} versus *clasp*^b_r, have little effect on the benchmarks in Table 4.12; this indifference does not apply to *smodels* and *smodels*$_r$, where restarts turn out to be counterproductive on the considered benchmarks.

4.5.3 Experiments on Projection Algorithm

Our third series of experiments illustrates the impact of computational strategies for enumerating projected answer sets. We consider *clasp* (version 1.2.0-RC3) using four different enumeration approaches. Two of them have already been investigated in the previous section, *clasp*^{a} running CDNL-ENUMERATION in Algorithm 4.6 on Page 81 and *clasp*^b_r running CDNL-RECORDING in Algorithm 4.5 on Page 78,[17] and are included here for comparison. The third mode (denoted by *clasp*^c_r) is also based on Algorithm 4.5 but records projections of solutions rather than entire solutions (as discussed at the beginning of Section 4.4.3). The fourth mode (denoted by *clasp*^{d}) runs CDNL-PROJECTION in Algorithm 4.7 on Page 87 to enumerate projected answer sets via a dedicated backtracking scheme. As with enumeration algorithms for entire solutions, restarts after the first solution are by default disabled with backtracking-based enumeration of projections (*clasp*^{d}), and uninfluenced when recording projections of solutions (*clasp*^c_r).

We also consider refinements of CDNL-PROJECTION differing in the way decisions are made in Line 34 and 39 of Algorithm 4.7, respectively. The variant *clasp*^{d}[h] uses *clasp*'s BerkMin-like decision heuristic to select σ_d in Line 34 (without sign selection); otherwise, an arbitrary unassigned entry of $nogood(bl)$ is picked. The variant *clasp*^{d}[p] utilizes the `--save-progress` option of *clasp* to direct the choice of σ_d in Line 39. Progress (or phase) saving [195] enforces sign selection for a picked variable according to the previously assigned truth value (if available) and thus guides search into similar directions as investigated earlier. Finally, the variant *clasp*^{d}[hp] combines the described features. The experiments were run on a 3.4GHz PC under Linux, each run restricted to 1000 seconds time and 1GB RAM.

We refrained from running other solvers than *clasp* because, to our knowledge, no other ASP solver supports dedicated enumeration of projected answer sets, and *clasp*^{a} as well as *clasp*^b_r already represent non-projecting enumeration approaches.[18] We also

[17] After finding a solution, (bounded) restarts are by default disabled with *clasp*^{a}, performing enumeration via backtracking, while *clasp*^{b}, recording solutions, sticks to its restart policy, as indicated by writing *clasp*^b_r.

[18] Any projecting or non-projecting enumeration approach can however be applied to the inputs used in

#Var	#Sol	$clasp^a$	$clasp_r^b$	$clasp_r^c$	$clasp^d$	$clasp^d$[h]	$clasp^d$[p]	$clasp^d$[hp]
1	11	100.38	>1000	0.01	0.01	0.01	0.01	0.01
2	110	100.38	>1000	0.01	0.01	0.01	0.01	0.01
3	990	100.38	>1000	0.05	0.07	0.06	0.07	0.07
4	7,920	100.38	>1000	0.60	0.35	0.34	0.35	0.35
5	55,440	100.38	>1000	9.08	1.67	1.68	1.61	1.67
6	332,640	100.38	>1000	281.05	6.34	6.32	6.50	6.34
7	1,663,200	100.38	>1000	>1000	20.63	20.17	21.04	20.39
8	6,652,800	100.38	>1000	>1000	49.97	51.20	50.10	49.18
9	19,958,400	100.38	>1000	>1000	88.77	88.73	89.63	91.18
10	39,916,800	100.38	>1000	>1000	114.17	119.36	119.12	114.82
11	39,916,800	100.38	>1000	>1000	114.30	113.92	116.80	118.83
∅		100.38	>1000	480.98	36.03	36.53	36.84	36.62

Table 4.13: Experiments enumerating projected answer sets: 11/11-pigeon-hole.

omitted trying decision heuristics preferably selecting output-relevant atoms (as used in the "Important First Decision Procedure" [130]) since such heuristics would require customizations of *clasp* offending the design goals postulated at the beginning of Section 4.4.

In Table 4.13 and 4.14, we investigate enumeration approaches relative to the proportion of atoms projected on. To this end, we consider two highly combinatorial benchmarks, the 11/11-pigeon-hole "problem" and the 15-queens puzzle. For both of them, we gradually increase the number of output-relevant atoms (in columns "#Var"), viz., the number of monitored pigeons or queens, respectively. The resulting numbers of projected answer sets are given in columns "#Sol"; the two rows above one marked with "∅" (displaying average runtimes of *clasp* variants) provide the total number of (unprojected) answer sets. We report runtimes in seconds; ">1000" stands for timeout, taken as 1000 seconds within averages. Note that the values in #Var and #Sol columns do not affect $clasp^a$ and $clasp_r^b$, which always (attempt to) enumerate all (unprojected) answer sets.

On the 11/11-pigeon-hole problem in Table 4.13, it is apparent that $clasp_r^b$ and $clasp_r^c$, persistently recording entire solutions or projected answer sets, respectively, do not scale. For the last instance solved by $clasp_r^c$, projecting on 6 out of 11 pigeons, the ratio of all answer sets to projections is 120. Furthermore, all variants of $clasp^d$ are faster than $clasp^a$, enumerating all answer sets, up to 9 out of 11 pigeons, at which point there are twice as many answer sets as distinct projections. For 10 and 11 pigeons, $clasp^a$ is slightly faster than $clasp^d$. In fact, $clasp^a$ saves some overhead by not distinguishing whether atoms in answer sets are output-relevant or not. However, there are no significant differences between the variants of $clasp^d$, given that the 11/11-pigeon-hole problem is fully symmetric and thus unaffected by heuristics.

On the 15-queens puzzle in Table 4.14, search becomes more important than on (satisfiable) pigeon-hole problems. Due to the reduced number of solutions, $clasp_r^b$ now completes in less than 1000 seconds, but it is still slower than backtracking-based enumeration schemes without persistent solution recording. We also note that $clasp_r^c$, recording projected answer sets, is the worst approach. In fact, its recorded projections consist of #Var entries each, while $clasp_r^b$ stores decision entries whose number decreases the

our experiments, given that "output" atoms are configured via the `#hide` and `#show` directives available in the input languages of ASP grounders like *gringo* [82, 91, 90] and *lparse* [211]. The observable difference between the enumeration of either entire answer sets or projections consists in how often subsets of (one or more) answer sets are produced, and in the runtime required to complete their enumeration.

#Var	#Sol	*clasp*[a]	*clasp*$_r$[b]	*clasp*$_r$[c]	*clasp*[d]	*clasp*[d][h]	*clasp*[d][p]	*clasp*[d][hp]
1	15	243.14	773.57	0.01	0.02	0.01	0.02	0.01
2	182	243.14	773.57	0.08	0.08	0.08	0.14	0.12
3	1,764	243.14	773.57	0.79	0.63	0.66	1.47	1.37
4	13,958	243.14	773.57	11.69	5.79	6.08	10.91	11.51
5	86,360	243.14	773.57	158.40	40.71	43.71	63.76	69.88
6	369,280	243.14	773.57	454.33	153.49	168.46	219.87	226.75
7	916,096	243.14	773.57	>1000	331.42	357.31	444.69	437.23
8	1,444,304	243.14	773.57	>1000	463.46	461.78	584.59	542.46
9	1,795,094	243.14	773.57	>1000	512.19	523.86	652.37	577.66
10	2,006,186	243.14	773.57	>1000	528.36	436.70	647.49	478.34
11	2,133,060	243.14	773.57	>1000	525.23	407.40	616.43	450.80
12	2,210,862	243.14	773.57	>1000	516.56	357.22	552.67	384.30
13	2,254,854	243.14	773.57	>1000	462.83	322.50	496.17	356.18
14	2,279,184	243.14	773.57	>1000	413.72	283.82	432.62	327.35
15	2,279,184	243.14	773.57	>1000	250.13	250.06	245.97	249.11
∅		243.14	773.57	641.69	280.31	241.31	331.28	274.20

Table 4.14: Experiments enumerating projected answer sets: 15-queens.

more solutions have been enumerated. Regarding the variants of *clasp*[d], we observe that the number of queens projected on and the resulting number of distinct projections do not affect their runtimes much beyond 7 queens. Rather, heuristic aspects of the search start to gain importance, and the variant *clasp*[d][h], which aims at placing the most critical queens first, has an edge. In contrast, stand-alone progress saving tends to misdirect search, as witnessed by *clasp*[d][p]. Furthermore, *clasp*[a], enumerating all 2,279,184 answer sets, becomes more efficient than the variants of *clasp*[d] from 7 queens on, where the ratio of all answer sets to projections is about 2.5. As on the 11/11-pigeon-hole problem, the reason is less overhead; in particular, *clasp*[a] does not even temporarily record any nogood for excluding repetitions. The reconvergence between *clasp*[a] and the variants of *clasp*[d] at 15 queens is by virtue of an implementation "trick": if the decision entry at level $bl+1$ in a solution (cf. Line 30–37 of Algorithm 4.7 on Page 87) involves a variable in P, then *clasp*[d] simply increments bl and backtracks to the greatest decision level at which some atom of P has been assigned; this shortcut permits unassigning fewer variables.

After inspecting two purely combinatorial problems, we now turn to more realistic benchmarks belonging to three different classes (cf. Table 4.15). The first one, *Clumpy*, deals with finding Hamiltonian cycles in clumpy graphs containing n clumps of n vertices each. For each value of n, we average over eleven randomly generated instances. Note that, due to high connectivity within clumps, clumpy graphs typically allow for a vast number of Hamiltonian cycles, but finding one is still difficult for systematic (chronological) backtracking methods (cf. [218]). In our experiments, we project Hamiltonian cycles to the edges connecting distinct clumps, thus reducing the number of distinguishable solutions by several orders of magnitude. The second class, *Repair*, stems from consistency checking of biological networks [80, 115]. Five categories, each containing thirty randomly generated instances, are distinguished by the number n, where $100 * n$ vertices are in a network. The task is to reestablish consistency by flipping observed variations (increase or decrease) of vertices. Solutions are then projected to the vertices whose variations are flipped in a repair, while discarding witnesses for consistency w.r.t. the repair. Given that there often are plenty witnesses, the number of distinct projections

Benchmark	n	*clasp*^{a}	*clasp*^b_r	*clasp*^c_r	*clasp*^{d}	*clasp*^{d}[h]	*clasp*^{d}[p]	*clasp*^{d}[hp]
Clumpy	08	204.50 (2)	468.48 (5)	0.02 (0)	0.02 (0)	0.02 (0)	0.02 (0)	0.02 (0)
	18	>1000 (11)	>1000 (11)	99.65 (1)	104.43 (1)	105.18 (1)	81.31 (0)	79.72 (0)
	20	>1000 (11)	>1000 (11)	255.04 (2)	254.80 (2)	313.22 (1)	219.05 (1)	118.95 (0)
	21	>1000 (11)	>1000 (11)	603.74 (6)	612.33 (6)	619.37 (6)	396.47 (4)	318.04 (3)
	22	>1000 (11)	>1000 (11)	144.64 (1)	266.72 (2)	275.54 (2)	410.98 (4)	321.07 (3)
∅ (Σ)		840.90 (46)	893.70 (49)	220.62 (10)	247.66 (11)	262.67 (10)	221.57 (9)	167.56 (6)
Repair	20	>1000 (30)	>1000 (30)	126.81 (0)	118.43 (0)	118.69 (0)	113.04 (0)	112.79 (0)
	25	>1000 (30)	>1000 (30)	232.57 (2)	223.07 (2)	223.37 (2)	217.17 (2)	216.22 (2)
	30	>1000 (30)	>1000 (30)	404.75 (6)	386.70 (5)	387.39 (5)	377.74 (5)	378.18 (5)
	35	>1000 (30)	>1000 (30)	322.10 (6)	312.76 (6)	312.72 (6)	306.93 (6)	306.67 (6)
	40	>1000 (30)	>1000 (30)	424.23 (7)	409.50 (7)	409.84 (7)	400.44 (7)	399.78 (7)
∅ (Σ)		>1000 (150)	>1000 (150)	302.09 (21)	290.09 (20)	290.40 (20)	283.06 (20)	282.73 (20)
Labyrinth	16	52.49 (0)	58.46 (1)	59.69 (1)	61.72 (1)	59.03 (1)	61.54 (1)	59.11 (1)
	17	165.15 (2)	162.60 (2)	198.32 (2)	220.13 (2)	196.83 (3)	220.26 (3)	198.25 (3)
	18	212.59 (2)	218.90 (2)	289.84 (4)	298.56 (3)	253.06 (3)	286.05 (3)	257.38 (3)
	19	238.24 (4)	241.26 (4)	260.63 (4)	266.96 (5)	245.83 (4)	264.68 (5)	250.90 (4)
	20	319.67 (5)	324.43 (5)	355.48 (6)	359.51 (7)	343.47 (6)	360.33 (7)	346.13 (6)
∅ (Σ)		197.63 (13)	201.13 (14)	232.79 (17)	241.38 (18)	219.64 (17)	238.57 (19)	222.35 (17)
∅ (Σ)		708.24 (209)	718.91 (213)	264.68 (48)	266.47 (49)	262.20 (47)	257.39 (48)	242.17 (43)

Table 4.15: Experiments enumerating projected answer sets: *Clumpy*, *Repair*, *Labyrinth*.

is several orders of magnitude smaller than the number of all answer sets. The third class, *Labyrinth*, considers a variation of Ravensburger's Labyrinth game on quadratic boards with n rows and n columns, each size n comprising twenty randomly generated configurations. The idea is that an avatar is guided from a starting to a goal position by moving the rows and columns of the board as well as the avatar itself, and projections disregard the moves of the avatar. It turns out that *Labyrinth* instances are pretty difficult to solve, and usually there are not many more answer sets than projections.

Table 4.15 shows average runtimes and numbers of timeouts (in parentheses) for *Clumpy*, *Repair*, and *Labyrinth* benchmarks. The rows marked with "∅(Σ)" provide the average runtime and sum of timeouts for each *clasp* variant over all instances of a benchmark class and in total (with benchmark classes weighted by their numbers of instances), respectively. For *Clumpy* and *Repair*, there are far too many answer sets to enumerate all of them with either *clasp*^{a} or *clasp*^b_r. Already on the smallest category of *Clumpy*, the approaches of the former cause timeouts, while enumerating projections with *clasp*^c_r or *clasp*^{d} is still unproblematic. On larger *Clumpy* categories, there is no clear winner among *clasp*^c_r and the variants of *clasp*^{d}, taking also into account that difficulty and number of projections vary significantly over instances. However, it appears that progress saving in *clasp*^{d}[p] and its combination with decision heuristic in *clasp*^{d}[hp] can be advantageous. On *Repair*, there are hardly any differences between the variants of *clasp*^{d}, and *clasp*^c_r, recording projected answer sets, is competitive too. Finally, on *Labyrinth*, *clasp*^{a} and *clasp*^b_r, both enumerating (unprojected) answer sets, have an edge on projecting enumeration approaches. This is unsurprising because there are not many more answer sets than projections for *Labyrinth* instances, and the disadvantages of enumerating projections are less dramatic than the advantages on other benchmarks. Regarding the variants of *clasp*^{d}, the use of a decision heuristic slightly promotes *clasp*^{d}[h] and *clasp*^{d}[hp], while stand-alone progress saving in *clasp*^{d}[p] does not help much on *Labyrinth*.

The summary given in the last row of Table 4.15 yields that the projecting enumeration approaches are close to each other, albeit *clasp*^{d}[hp] has a small advantage. The

latter suggests that enumeration may benefit from the incorporation of search techniques, such as a decision heuristic or progress saving. Their usefulness, however, depends on a problem at hand, and thus fine-tuning is needed. Importantly, the enumeration of all projections is sometimes still possible when there are far too many (unprojected) answer sets to enumerate all of them, which may be crucial for the feasibility of applications.

4.6 Related Work

Our algorithms for conflict-driven ASP solving borrow and extend state-of-the-art techniques from the area of SAT solving [21]. Their global search pattern is similar to CDCL with First-UIP scheme, developed around a decade ago [179, 185, 219] and nowadays quasi standard for industrial SAT solving (cf. [56, 203, 183, 48, 20, 7, 41, 178]). While classical DPLL-style procedures are polynomially equivalent to tree-like resolution (cf. [19]), CDCL (with unlimited restarts) amounts to general resolution [18, 197] and is thus strictly more powerful than DPLL.

Beyond the basic decision procedure in Section 4.3, we have presented algorithms for the enumeration of answer sets that harness conflict-driven learning and backjumping according to the First-UIP scheme. Solution recording, as described in Section 4.4.1, is closely related to blocking clauses [151, 146, 184, 155], recorded in conflict-driven enumeration procedures for (classical) models of propositional theories. Approaches aiming at the compaction of (enumerated) solutions' representation can be found in [146, 184]. Unlike this, our enumeration algorithms in Section 4.4.2 and 4.4.3 apply dedicated backtracking schemes to abolish the need of (persistently) recording solutions. Moreover, dedicated backtracking precludes the enumeration of duplicates, while the deletion of non-asserting blocking clauses proposed in [155] risks the repetition of solutions. Applications of enumeration (with and without projection) arise, e.g., in combinatorial mathematics [147], itemset mining [142], model checking [130], predicate abstraction [155], probabilistic reasoning [10], and systems biology [115].

SAT-based enumerators utilizing conflict-driven learning, but not blocking clauses, include *relsat* [17, 16], which relies on the Last-UIP scheme (cf. [219]), and the "Important First Decision Procedure" [130]. Like our enumeration algorithm for projected answer sets, CDNL-PROJECTION, the "Important First Decision Procedure" mainly aims at enumerating projections of solutions, including projection via identity as a special case. The major difference between both approaches is that the "Important First Decision Procedure" restricts decision heuristics a priori, which deteriorates proof complexity [143], while CDNL-PROJECTION reorganizes decisions a posteriori after solutions have been obtained. Furthermore, the "Important First Decision Procedure" restricts backjumping according to the First-UIP scheme to "non-important" variables and, otherwise, applies a conflict resolution and backjumping scheme similar to the one of *relsat*; it also records non-asserting conflict clauses, which can be retrieved in the presence of flipped decision entries lacking antecedents. Our enumeration algorithms preclude the latter, given that they do not analyze conflicts due to flipped decision entries and rather apply systematic backtracking to recover from them. Also note that our enumeration algorithms are devised not to incur any computational overhead (in comparison to the basic decision procedure they extend) a priori, that is, before any answer set is found, which distinguishes them from #SAT methods that utilize conflict-driven learning and backjumping according to the First-UIP scheme (cf. [128]).

Given that answer sets are determined by atoms, native ASP solvers *dlv* [158], *smodels* [209], and *smodels*$_{cc}$ [218] are (logically) restricted to assignments over atoms. As shown in Section 3.4, this yields an exponential separation, already on tight programs, to solvers that in addition assign and make decisions on rule bodies. To our knowledge, *clasp* [95] and *nomore++* [3] are the only ASP solvers deliberately including rule bodies in assignments. Interestingly, Conjunctive Normal Form (CNF) conversions applied by SAT-based ASP solvers like *assat* [167], *cmodels* [123], and *sag* [168] also introduce auxiliary variables for rule bodies to prevent an exponential blow-up (cf. [9]). Although such variables can then be exploited by underlying SAT solvers, their motivation is more by need than by design. As there is not yet a consensus on how to represent the constraints induced by a logic program, in Section 4.1, we have used nogoods to express conditions for (unit) propagation, thus separating semantics from syntactic representations.

Like *clasp*, SAT-based ASP solvers may exploit conflict-driven learning in the search for answer set candidates (cf. Section 3.2.3), accomplished by underlying SAT solvers. However, their integration of unfounded set checking is much more loose than in our approach, where it is part of propagation and extracted loop nogoods serve as antecedents, as described in Section 4.3.2. The eager translations offered by *lp2sat* [139] and *lp2diff* [141] embed unfounded set handling into SAT or difference logic [188], respectively, by encoding level mappings [136, 187] w.r.t. non-tight programs, in subquadratic (*lp2sat*) or even linear (*lp2diff*) space. To our knowledge, the only native ASP solver other than *clasp* that implements conflict-driven learning is *smodels*$_{cc}$,[19] while traditional ASP solvers like *dlv*, *smodels*, and *nomore++* perform DPLL-style search. For enabling conflict resolution, *smodels*$_{cc}$ takes an algorithmic approach, monitoring applications of *smodels*' propagation rules (cf. Section 3.2.2) to on-the-fly build an implication graph [179, 219, 18] as a representation of antecedents. In contrast to *smodels*$_{cc}$, our semantic framework allows us to view deterministic inferences as unit propagation on nogoods, which directly provide antecedents.

Unlike SAT-based ASP solvers, native ones tightly integrate unfounded set checking into their propagation routines, and virtually all ASP solvers exploit strongly connected components of dependency graphs to confine work to necessary parts. While the unfounded set detection procedures of *dlv* [32] and *smodels* [209] identify greatest unfounded sets [216], those of *clasp*, detailed in Section 4.3.3, and *nomore++* [4] aim at identifying small (non-empty) unfounded sets and return them as soon as they are detected. The main motive for this is to reduce overlaps between (unit) propagation and unfounded set checking. Another difference between unfounded set detection procedures is that *dlv* and *nomore++* use a flag "must-be-true" to indicate (logically true) atoms whose acyclic derivability is uncertain, for which purpose *smodels* and *clasp* exploit source pointers [209]. The advantage of source pointers is that they need not be updated upon backtracking or backjumping, respectively, while "true" may have to be turned back into "must-be-true." Thus, source pointers can be regarded as the unfounded set detection counterpart of watched literals [185, 56, 203, 20], a data structure for implementing unit propagation lazily, popular due to invariance under backtracking/backjumping and economic cache utilization.

While several approaches [167, 123, 168, 4, 102] admit limiting the consideration of unfounded sets to loops, *clasp* does not make sure that a detected non-empty unfounded

[19] The solver *minisat(id)* [177] supports inductive definitions on top of propositional theories. Inductive definitions are closely related to logic programs, yet involve a "totality" condition not shared by the latter.

set is a loop (yet it contains one), and it is an interesting open question whether a strict limitation to loops would be advantageous. Furthermore, we note that the "unidirectional" unfounded set handling in native ASP solvers, not realizing full unit propagation via loop formulas (cf. Section 3.2), has also been recognized in [111, 110]. As discussed in Section 4.3.4, this may lead to "trivial" (First-)UIPs without performing any conflict resolution step. Unfortunately, the approach to remedy this peculiarity suggested in [33, 34] is computationally too complex (quadratic) to be beneficial in practice, and it is open whether the same effect can be achieved by more economic techniques.

In addition to its fundamental algorithms, which have been presented here, *clasp* supports various extended functionalities (cf. [97]). It offers advanced preprocessing of logic programs [96] and their induced constraints [54]. Techniques for the native treatment of extended rules [209] are presented in [84], including an unfounded set detection procedure extending the one in Algorithm 4.3. Several systems implement particular features on top of *clasp*: the disjunctive ASP solver *claspD* [52] internally couples two *clasp* engines, *clingcon* [105] embeds the *gecode* constraint library[20] into *clasp*'s propagation routine to deal with non-Boolean variables, *iclingo* [83] exploits *clasp*'s incremental interface to solve series of problems over increasing horizons [36, 55], and the parallel ASP solver *claspar* [59, 207, 88] augments *clasp* with a communication module to enable message passing between distributed solver instances.

4.7 Discussion

We have provided a uniform approach to conflict-driven ASP solving based on nogoods, utilized to express constraints induced by a normal program. While Chapter 3 has laid proof-theoretic foundations by providing inference rules, the corresponding nogoods now allow us to view deterministic inferences as applications of unit propagation. This bridges the semantic gap between ASP and SAT solving and enables direct technology transfers between both areas, without the need of translation. In fact, certain features of computational (search) problems, such as least fixpoints, recursion, and transitivity, are inherently supported by the semantics of ASP and can be represented succinctly by logic programs (cf. [206, 176, 186, 136, 164]). In order to keep this succinctness, we did not defer to CNF conversions of logic programs and SAT solving, but devised modern ASP solving procedures that integrate and extend state-of-the-art techniques from SAT solving.

In contrast to SAT, where nogoods are expatiated by clauses, logic programs induce further implicit constraints, given by loop nogoods. Though inherently present, these nogoods need only be inspected when they are antecedents or violated. Based on this perception, we have developed a conflict-driven approach to ASP solving in which loop nogoods are tested via a dedicated unfounded set checking routine. However, our approach favors unit propagation on nogoods stemming from completion or previous conflicts over unfounded set computations. This makes sure that inspected loop nogoods are "1-empowering" [196] and supplement the available constraints.

Beyond a decision procedure, we have presented dedicated backtracking-based algorithms for the repetition-free enumeration of answer sets and their projections to a subvocabulary, respectively, in polynomial space. Our enumeration algorithms harness conflict-driven learning and backjumping according to the First-UIP scheme, and they do not incur computational overhead a priori, that is, before any answer set is found. Ex-

[20] Available at `http://www.gecode.org`.

tending modern ASP (or SAT) solving procedures to repetition-free enumeration, while still running in polynomial space, is non-trivial: to the best of our knowledge, the algorithms presented here are the first non-intrusive extensions. That is, our algorithms can be utilized by solvers that aim at combining high performance on decision problems with the flexibility to also handle applications relying on enumeration.

Our approach is implemented in the award-winning Boolean constraint solver *clasp*, which augments the fundamental algorithms presented here with many additional functionalities (cf. Section 4.5). Its attractiveness is witnessed by an increasing number of applications relying on *clasp* or its derivatives as reasoning engines, e.g., [182, 22, 134, 152, 213, 115]. Although we have confined ourselves to the presentation of principles and omitted details on extended rules (cf. [84]) or the actual implementation of *clasp*, we believe that a comprehensive overview of the mechanisms underlying *clasp* may be useful for an audience interested in the design and operation of modern ASP solvers. Last but not least, let us note that *clasp* is **freely available** as an **open source** package at [198], and thus it is instantly accessible for users as well as contributors.

Chapter 5

Conclusions

This thesis provided semantic, proof-theoretic, and algorithmic foundations for modern ASP solving approaches. In Chapter 2, we extended the concept of an unfounded set [216] to assignments over atoms as well as rule bodies occurring in a normal logic program. Tableau rules working on such twofold assignments have been presented in Chapter 3. We have seen that they allow us to characterize a variety of familiar logic programming concepts as well as existing ASP solvers in a uniform setting. Notably, exponential separations are obtained when case analyses are restricted to only either atoms or rules bodies. We extended our approach to the richer setting of logic programs with aggregates by also including them in assignments, and we demonstrated that exponential separations due to case analyses are also possible for aggregates. The constraint-based approach pursued in Chapter 4 benefits from assignments including rule bodies, as they admit compact characterizations of answer sets in terms of nogoods and of deterministic inferences by unit propagation. On this basis, we developed conflict-driven procedures for the decision problem of answer set existence as well as solution enumeration. The main particularity of answer set computation is the detection of unfounded sets, for which purpose we provided an elaborate algorithm exploiting source pointers [209]. Finally, our enumeration algorithms combine conflict-driven learning and backjumping according to the First-UIP scheme [179, 219] with dedicated backtracking in a non-intrusive way, admitting the repetition-free enumeration of answer sets as well as their projections in polynomial space.

In more detail, the tableau rules presented in Chapter 3 focus on fundamental inference steps w.r.t. the syntax of logic programs. Our framework makes more involved characterizations (like the ones in [3, 32, 64, 122, 154, 169, 209, 218]) obsolete, as they are subsumed by particular tableau calculi. In fact, our tableaux provide a uniform view on different ASP solving approaches; this also includes the inferences of SAT-based ASP solvers [123, 167, 168]. In accord with unfounded set checking techniques (w.r.t. total assignments) of the latter, we have presented sound and complete tableau calculi that limit (sophisticated) unfounded set checks to loops [167], already w.r.t. partial assignments. It would even be possible to go beyond this by considering *elementary* loops [101, 102, 103, 109] only (which has been omitted here for brevity). In the context of logic programs with aggregates, for which there is no agreed unfounded set definition,[1] our concept $sup_{\mathbf{A}}(\Pi, S, S)$, exploited by the generic tableau rules $U{\uparrow}$ and $U{\downarrow}$, intrinsically characterizes unfounded sets. As this characterization does not rely on translation

[1] The unfounded set concept proposed in [65] fails to reproduce the general aggregate semantics in [69].

to propositional theories [70] and is not limited to particular aggregates [170], direct definitions of unfounded sets, loops, and loop formulas for logic programs with aggregates may be oriented at our methodology.

The algorithms presented in Chapter 4 integrate ASP-specific inferences with modern conflict-driven search patterns [41, 178]. To this end, we provided a constraint-oriented semantic framework in which all inferences from logic programs are characterized by unit propagation. Importantly, our choice to express constraints in terms of nogoods does not aim at suggesting any particular syntactic representation of the constraints induced by a logic program. This especially applies to loop nogoods justifying inferences due to unfounded sets, as their succinct representation given by a logic program avoids an exponential blow-up, faced when expatiating such conditions by clauses [164]. Rather, we devised a dedicated unfounded set detection algorithm that determines loop nogoods on demand, that is, when they are antecedents or violated; such "applicable" loop nogoods are then (temporarily) recorded to make them easily accessible for conflict resolution. Note that our constraint-based characterization of answer sets in terms of nogoods puts search in ASP on the same foundation as in neighboring areas, most directly, SAT [21] and Pseudo-Boolean (PB) [50, 202] satisfiability. In fact, unfounded set checking in ASP solving can be viewed as a particular form of theory propagation, as performed in solvers for Satisfiability Modulo Theories (SMT) [15].

The success and versatility of our methodology are demonstrated by *clasp*, the current state-of-the-art ASP solver that implements the approach presented here, winning first places in the 2009 competitions for ASP[2] [47], SAT[3], and PB[4]. To our knowledge, *clasp* is still the only ASP solver that has been genuinely developed for conflict-driven ASP solving, thus going beyond SAT-based approaches [123, 167, 168]. Indeed, the elaborate features of *clasp*, such as preprocessing [96], native treatment of extended rules [84], and dedicated backtracking schemes for enumeration [94, 98], are outstanding. In this regard, we would like to stress that starting from scratch in developing both the theoretical fundament and the practical implementation[5] of a modern Boolean constraint solver for ASP was indispensable to achieve the results obtained so far. Notably, the rich yet easy modeling languages of ASP [82, 158, 211] facilitate the formulation of application problems, which can then be solved by using *clasp* (or other ASP solvers). For the future, it would be desirable that modeling capacities were combined with broader solver development efforts, e.g., by extending high-performance SAT solvers to ASP solving rather than distorting logic programs to fit the needs of plain SAT solvers.

This thesis (and previous work it is based on) abolished preexisting barriers between separate areas of Boolean constraint solving by demonstrating how conflict-driven learning can be extended transparently from SAT to ASP. We thus made a step towards combining elevated expressiveness (of ASP) with powerful search techniques (from SAT) in order to likewise model and solve computationally complex problems effectively. However, most of the way along this road is still unexplored, and only the future can tell how it continues. To give an idea, though, we note that ongoing work meanwhile includes the auto-configuration of *clasp* [87], its application to optimization problems [85, 86], parallelization [59, 88, 207], and conceptual extensions for solving logic programs capturing

[2] See `http://dtai.cs.kuleuven.be/events/ASP-competition`.

[3] See `http://www.satcompetition.org`.

[4] See `http://www.cril.univ-artois.fr/PB09`.

[5] As already mentioned, the author is deeply grateful to Benjamin Kaufmann and all other contributors for doing excellent and invaluable work in developing and constantly improving *clasp*.

problems located at the second level of the polynomial time hierarchy (prototypically performed in *claspD* [52]).

As a further step, integrations of ASP and Constraint Programming (CP) [201] or SMT, respectively, have recently been proposed [11, 105, 137, 181] for dealing with non-Boolean variables and sophisticated constraints. Up to now, however, such hybrid frameworks are usually conceived as Boolean constraint solving augmented with separate components, rather than instances of a general approach with a specialized Boolean core. A plausible explanation could be that, to our knowledge, the success of conflict-driven learning was so far largely limited to propositional (two-valued) formalisms (cf. [149, 150]), and comparably effective extensions are still lacking. However, we speculate that viewing different constraint languages from a unified perspective might allow for overcoming such difficulties in the future. Ultimately, (largely) automated problem solving likewise requires expressive knowledge representation capacities and powerful solving methods. Although an "omnipotent" paradigm for modeling and solving any kind of computationally complex problem appears to be science fiction, combined efforts and cross-fertilizations between diverse areas of declarative programming may foster steady steps towards this superordinate goal.

Appendix A

Examples

This appendix provides detailed traces of the enumeration algorithms in Section 4.4, separately for examples.

A.1 Example 4.10

Table A.1, A.2, and A.3 show the complete trace of a computation of all answer sets of Π_{11} from Example 4.10 with CDNL-RECORDING($\Pi_{11}, 0$). The first column gives the value of dl, viz., the current decision level, at which particular instructions are performed. Concerned entries and nogoods are shown in the columns headed by $\mathbf{A}$ and δ, respectively. The "Info" column displays additional information about answer sets or nogoods, and "Line" indicates at which line of Algorithm 4.5 particular contents of $\mathbf{A}$ and/or some nogood δ are inspected.

The reading of Table A.1, A.2, and A.3 is as follows. A block starting with a particular value of dl $(1, 2, 3, \dots)$ remains intact, i.e., the entries σ in $\mathbf{A}$ remain assigned at $dlevel(\sigma) = dl$ until a new block with the same value of dl begins. When such a new block overrides a previous one with the same value of dl, entries σ that stay assigned at $dlevel(\sigma) = dl$ are repeated without indicating any line of Algorithm 4.5, while a line is given for each newly assigned entry. Regarding the meanings of line numbers, 19 indicates that a decision entry is assigned, and 5 that an implied entry is inserted into $\mathbf{A}$, whose antecedent δ belongs to the set of nogoods reported as info. (Implied entries are assigned in "random order," as the existence of some antecedent δ is sufficient for conflict resolution to be well-defined.) A conflict, viz., a nogood contained in $\mathbf{A}$, is encountered in Line 6, and the result of CONFLICTANALYSIS in Line 8, a nogood δ to be recorded in ∇ and an assertion level dl, is provided in this case. The set of true atoms in a solution for $\Delta_{\Pi_{11}} \cup \Lambda_{\Pi_{11}}$, printed in Line 12, is an answer set of Π_{11}, which is shown underlined (as info); we also indicate the respective entries via underlining when a partial assignment is extended to a solution. An associated solution-suppressing nogood, recorded in ∇, gives rise to a conflict in the row below an answer set. Finally, note that decision level 0 is highlighted in bold face since a conflict at decision level 0 provides the termination condition in Line 7 of Algorithm 4.5.

Overall, Table A.1, A.2, and A.3 show how the answer sets $\{y, a, b\}$, $\{y, a, c\}$, $\{x, a, b, c\}$, $\{z, a, b\}$, and $\{z, a, c\}$ of Π_{11} are enumerated with CDNL-RECORDING($\Pi_{11}, 0$).

dl	**A**	δ	Info	Line
1	$\boldsymbol{T}y$			19
	$\boldsymbol{F}\{not\ y, not\ z\}$	$\{\boldsymbol{T}\{not\ y, not\ z\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x, not\ y\}$	$\{\boldsymbol{T}\{not\ x, not\ y\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	5
2	$\boldsymbol{T}a$			19
3	$\boldsymbol{T}x$			19
	$\boldsymbol{F}\{not\ x, not\ z\}$	$\{\boldsymbol{T}\{not\ x, not\ z\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{x, not\ b\}$	$\{\boldsymbol{T}y, \boldsymbol{F}\{x, not\ b\}, \boldsymbol{F}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}b$	$\{\boldsymbol{T}\{x, not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{F}b, \boldsymbol{T}\{x\}\}$	$\Delta_{\Pi_{11}}$	5
		$\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	6
		$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	dl=1	8
1	$\underline{\boldsymbol{T}y}$			
	$\boldsymbol{F}\{not\ y, not\ z\}$	$\{\boldsymbol{T}\{not\ y, not\ z\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{not\ x, not\ y\}$	$\{\boldsymbol{T}\{not\ x, not\ y\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	∇	5
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{F}x, \boldsymbol{T}\{b, c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{T}\{x, not\ b\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ c\}$	$\{\boldsymbol{T}\{x, not\ c\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ z\}$	$\{\boldsymbol{T}\{x, not\ z\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}z$	$\{\boldsymbol{T}z, \boldsymbol{F}\{x, not\ c\}, \boldsymbol{F}\{not\ x, not\ y\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x, not\ z\}$	$\{\boldsymbol{T}y, \boldsymbol{F}\{x, not\ b\}, \boldsymbol{F}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x\}$	$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}a}$	$\{\boldsymbol{F}a, \boldsymbol{T}\{not\ x\}\}$	$\Delta_{\Pi_{11}}$	5
2	$\underline{\boldsymbol{T}b}$			19
	$\boldsymbol{T}\{not\ c\}$	$\{\boldsymbol{T}b, \boldsymbol{F}\{x\}, \boldsymbol{F}\{not\ c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ b\}$	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}c$	$\{\boldsymbol{T}\{not\ c\}, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{y, a, b\}}$	12
		$\{\boldsymbol{T}y, \boldsymbol{F}x, \boldsymbol{F}z, \boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}c\}$	∇	6
		$\{\boldsymbol{T}y, \boldsymbol{F}x, \boldsymbol{F}z, \boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}\{x\}\}$	dl=1	8
1	$\underline{\boldsymbol{T}y}$			
	$\boldsymbol{F}\{not\ y, not\ z\}$	$\{\boldsymbol{T}\{not\ y, not\ z\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{not\ x, not\ y\}$	$\{\boldsymbol{T}\{not\ x, not\ y\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	∇	
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{F}x, \boldsymbol{T}\{b, c\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{T}\{x, not\ b\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{x, not\ c\}$	$\{\boldsymbol{T}\{x, not\ c\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{x, not\ z\}$	$\{\boldsymbol{T}\{x, not\ z\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}z$	$\{\boldsymbol{T}z, \boldsymbol{F}\{x, not\ c\}, \boldsymbol{F}\{not\ x, not\ y\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{T}\{not\ x, not\ z\}$	$\{\boldsymbol{T}y, \boldsymbol{F}\{x, not\ b\}, \boldsymbol{F}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{T}\{not\ x\}$	$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	
	$\underline{\boldsymbol{T}a}$	$\{\boldsymbol{F}a, \boldsymbol{T}\{not\ x\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}b$	$\{\boldsymbol{T}y, \boldsymbol{F}x, \boldsymbol{F}z, \boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}\{x\}\}$	∇	5
	$\boldsymbol{F}\{not\ c\}$	$\{\boldsymbol{F}b, \boldsymbol{T}\{not\ c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ b\}$	$\{\boldsymbol{F}\{not\ b\}, \boldsymbol{F}b\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}c}$	$\{\boldsymbol{F}\{not\ c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{y, a, c\}}$	12
		$\{\boldsymbol{T}y, \boldsymbol{F}x, \boldsymbol{F}z, \boldsymbol{T}a, \boldsymbol{F}b, \boldsymbol{T}c\}$	∇	6
		$\{\boldsymbol{T}y\}$	dl=$\mathbf{0}$	8

Table A.1: First part of a computation of all answer sets with CDNL-RECORDING($\Pi_{11}, 0$).

dl	$\mathbf{A}$	δ	Info	Line
0	$\boldsymbol{F}y$	$\{\boldsymbol{T}y\}$	∇	5
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{x, not\ b\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x, not\ z\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
1	$\underline{\boldsymbol{T}b}$			19
	$\boldsymbol{F}\{not\ b\}$	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	5
2	$\underline{\boldsymbol{T}c}$			19
	$\boldsymbol{F}\{not\ c\}$	$\{\boldsymbol{T}\{not\ c\}, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ c\}$	$\{\boldsymbol{T}\{x, not\ c\}, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{b, c\}$	$\{\boldsymbol{F}\{b, c\}, \boldsymbol{T}b, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{x\}$	$\{\boldsymbol{T}b, \boldsymbol{F}\{x\}, \boldsymbol{F}\{not\ c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{x, not\ z\}$	$\{\boldsymbol{T}c, \boldsymbol{F}\{x, not\ z\}, \boldsymbol{F}\{not\ b\}\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}a}$	$\{\boldsymbol{F}a, \boldsymbol{T}\{x\}\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}x}$	$\{\boldsymbol{F}x, \boldsymbol{T}\{b, c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x\}$	$\{\boldsymbol{T}\{not\ x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x, not\ y\}$	$\{\boldsymbol{T}\{not\ x, not\ y\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}z$	$\{\boldsymbol{T}\{x, not\ z\}, \boldsymbol{T}z\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ y, not\ z\}$	$\{\boldsymbol{F}\{not\ y, not\ z\}, \boldsymbol{F}y, \boldsymbol{F}z\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{x, a, b, c\}}$	12
		$\{\boldsymbol{F}y, \boldsymbol{T}b, \boldsymbol{T}c, \boldsymbol{T}a, \boldsymbol{T}x, \boldsymbol{F}z\}$	∇	6
		$\{\boldsymbol{F}y, \boldsymbol{T}b, \boldsymbol{T}c, \boldsymbol{F}\{not\ b\}\}$	dl=1	8
1	$\boldsymbol{T}b$			
	$\boldsymbol{F}\{not\ b\}$	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}c$	$\{\boldsymbol{F}y, \boldsymbol{T}b, \boldsymbol{T}c, \boldsymbol{F}\{not\ b\}\}$	∇	5
	$\boldsymbol{T}\{not\ c\}$	$\{\boldsymbol{F}\{not\ c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{T}\{b, c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ z\}$	$\{\boldsymbol{F}c, \boldsymbol{T}\{x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
2	$\boldsymbol{F}a$			19
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{F}a, \boldsymbol{T}\{x\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x\}$	$\{\boldsymbol{F}a, \boldsymbol{T}\{not\ x\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}x$	$\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
		$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	6
		$\{\boldsymbol{F}a\}$	dl=**0**	8
0	$\boldsymbol{F}y$	$\{\boldsymbol{T}y\}$	∇	
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{x, not\ b\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{not\ x, not\ z\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{T}a$	$\{\boldsymbol{F}a\}$	∇	5
1	$\boldsymbol{F}z$			19
	$\boldsymbol{T}\{not\ y, not\ z\}$	$\{\boldsymbol{F}\{not\ y, not\ z\}, \boldsymbol{F}y, \boldsymbol{F}z\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}x$	$\{\boldsymbol{F}x, \boldsymbol{T}\{not\ y, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{x\}$	$\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{x, not\ z\}$	$\{\boldsymbol{F}\{x, not\ z\}, \boldsymbol{T}x, \boldsymbol{F}z\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}b$	$\{\boldsymbol{F}b, \boldsymbol{T}\{x\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}c$	$\{\boldsymbol{F}c, \boldsymbol{T}\{x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
		$\{\boldsymbol{F}y, \boldsymbol{T}b, \boldsymbol{T}c, \boldsymbol{T}a, \boldsymbol{T}x, \boldsymbol{F}z\}$	∇	6
		$\{\boldsymbol{F}y, \boldsymbol{T}a, \boldsymbol{F}z\}$	dl=**0**	8

Table A.2: Second part of a computation of all answer sets with CDNL-RECORDING($\Pi_{11}, 0$).

dl	**A**	δ	Info	Line
0	$\boldsymbol{F}y$	$\{\boldsymbol{T}y\}$	∇	
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{x, not\ b\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{not\ x, not\ z\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	
	$\underline{\boldsymbol{T}a}$	$\{\boldsymbol{F}a\}$	∇	
	$\underline{\boldsymbol{T}z}$	$\{\boldsymbol{F}y, \boldsymbol{T}a, \boldsymbol{F}z\}$	∇	5
	$\boldsymbol{F}\{not\ y, not\ z\}$	$\{\boldsymbol{T}\{not\ y, not\ z\}, \boldsymbol{T}z\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ z\}$	$\{\boldsymbol{T}\{x, not\ z\}, \boldsymbol{T}z\}$	$\Delta_{\Pi_{11}}$	5
1	$\underline{\boldsymbol{T}b}$			19
	$\boldsymbol{F}\{not\ b\}$	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}c$	$\{\boldsymbol{T}c, \boldsymbol{F}\{x, not\ z\}, \boldsymbol{F}\{not\ b\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ c\}$	$\{\boldsymbol{F}\{not\ c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{T}\{b, c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{not\ y, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ c\}$	$\{\boldsymbol{T}\{x, not\ c\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x\}$	$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x, not\ y\}$	$\{\boldsymbol{F}\{not\ x, not\ y\}, \boldsymbol{F}x, \boldsymbol{F}y\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{z, a, b\}}$	12
		$\{\boldsymbol{F}y, \boldsymbol{T}a, \boldsymbol{T}z, \boldsymbol{T}b, \boldsymbol{F}c, \boldsymbol{F}x\}$	∇	6
		$\{\boldsymbol{F}y, \boldsymbol{T}a, \boldsymbol{T}z, \boldsymbol{T}b, \boldsymbol{F}\{not\ y, not\ z\}, \boldsymbol{F}\{x, not\ z\}\}$	dl=**0**	8
0	$\boldsymbol{F}y$	$\{\boldsymbol{T}y\}$	∇	
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{x, not\ b\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{not\ x, not\ z\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	
	$\underline{\boldsymbol{T}a}$	$\{\boldsymbol{F}a\}$	∇	
	$\underline{\boldsymbol{T}z}$	$\{\boldsymbol{F}y, \boldsymbol{T}a, \boldsymbol{F}z\}$	∇	
	$\boldsymbol{F}\{not\ y, not\ z\}$	$\{\boldsymbol{T}\{not\ y, not\ z\}, \boldsymbol{T}z\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{x, not\ z\}$	$\{\boldsymbol{T}\{x, not\ z\}, \boldsymbol{T}z\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}b$	$\{\boldsymbol{F}y, \boldsymbol{T}a, \boldsymbol{T}z, \boldsymbol{T}b, \boldsymbol{F}\{not\ y, not\ z\}, \boldsymbol{F}\{x, not\ z\}\}$	∇	5
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{T}\{b, c\}, \boldsymbol{F}b\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ b\}$	$\{\boldsymbol{F}\{not\ b\}, \boldsymbol{F}b\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}c}$	$\{\boldsymbol{F}c, \boldsymbol{T}\{not\ b\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ c\}$	$\{\boldsymbol{F}b, \boldsymbol{T}\{not\ c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{F}b, \boldsymbol{T}\{x\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}x$	$\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ c\}$	$\{\boldsymbol{T}\{x, not\ c\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x\}$	$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x, not\ y\}$	$\{\boldsymbol{F}\{not\ x, not\ y\}, \boldsymbol{F}x, \boldsymbol{F}y\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{z, a, c\}}$	12
		$\{\boldsymbol{F}y, \boldsymbol{T}a, \boldsymbol{T}z, \boldsymbol{F}b, \boldsymbol{T}c, \boldsymbol{F}x\}$	∇	6

Table A.3: Third part of a computation of all answer sets with CDNL-RECORDING($\Pi_{11}, 0$).

A.2 Example 4.11

Table A.4, A.5, and A.6 show the complete trace of a computation of all answer sets of Π_{11} from Example 4.10 with CDNL-ENUMERATION($\Pi_{11}, 0$). The structure and reading of theses tables are as explained in Appendix A.1, except for the following differences:

- Decision entries are inserted into $\mathbf{A}$ in Line 30 (not in Line 19) of Algorithm 4.6.
- A result of CONFLICTANALYSIS is obtained in Line 9 (not in Line 8) of Algorithm 4.6.
- The set of true atoms in a solution for $\Delta_{\Pi_{11}} \cup \Lambda_{\Pi_{11}}$ is printed in Line 19 (not in Line 12) of Algorithm 4.6.
- After encountering a solution for $\Delta_{\Pi_{11}} \cup \Lambda_{\Pi_{11}}$, the complement of a former decision entry is assigned in Line 25 of Algorithm 4.6, and we highlight the backtracking level bl in bold face.
- In reaction to a conflict at the backtracking level (beyond 0), its decision entry is flipped in Line 17 of Algorithm 4.6, while CONFLICTANALYSIS is not invoked.

A.3 Example 4.12

Table A.7 and A.8 show the complete trace of a computation of all projections of answer sets of Π_{11} from Example 4.10 to $\{a, b, c\}$ with CDNL-PROJECTION($\Pi_{11}, \{a, b, c\}, 0$). The structure and reading of theses tables are as explained in Appendix A.1, except for the following differences:

- Decision entries are inserted into $\mathbf{A}$ in Line 41 (not in Line 19) of Algorithm 4.7.
- A result of CONFLICTANALYSIS is obtained in Line 9 (not in Line 8) of Algorithm 4.7.
- The set of true atoms in the projection of a solution for $\Delta_{\Pi_{11}} \cup \Lambda_{\Pi_{11}}$ to $\{a, b, c\}$ is printed in Line 20 (not in Line 12) of Algorithm 4.7. Furthermore, only the true entries over the "output" atoms $\{a, b, c\}$ are indicated via underlining when a partial assignment is extended to a solution.
- The projection of a solution may be (temporarily) recorded in ∇ as $nogood(bl)$ in Line 32 of Algorithm 4.7, in which case some of its entries is reassigned as $decision(bl)$ in Line 37, and we highlight the backtracking level bl in bold face.
- A solution-suppressing nogood, $nogood(bl)$, may be deleted in Line 24 (or 14) of Algorithm 4.7, which is indicated by striking the respective nogood. The associated flipping of a former decision entry is performed in Line 28 (or 18).

dl	$\mathbf{A}$	δ	Info	Line
1	$\boldsymbol{T}y$			30
	$\boldsymbol{F}\{not\ y, not\ z\}$	$\{\boldsymbol{T}\{not\ y, not\ z\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x, not\ y\}$	$\{\boldsymbol{T}\{not\ x, not\ y\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	5
2	$\boldsymbol{T}a$			30
3	$\boldsymbol{T}x$			30
	$\boldsymbol{F}\{not\ x, not\ z\}$	$\{\boldsymbol{T}\{not\ x, not\ z\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{x, not\ b\}$	$\{\boldsymbol{T}y, \boldsymbol{F}\{x, not\ b\}, \boldsymbol{F}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}b$	$\{\boldsymbol{T}\{x, not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{F}b, \boldsymbol{T}\{x\}\}$	$\Delta_{\Pi_{11}}$	5
		$\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	6
		$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	dl=1	9
1	$\underline{\boldsymbol{T}y}$			
	$\boldsymbol{F}\{not\ y, not\ z\}$	$\{\boldsymbol{T}\{not\ y, not\ z\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{not\ x, not\ y\}$	$\{\boldsymbol{T}\{not\ x, not\ y\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	∇	5
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{F}x, \boldsymbol{T}\{b, c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{T}\{x, not\ b\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ c\}$	$\{\boldsymbol{T}\{x, not\ c\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ z\}$	$\{\boldsymbol{T}\{x, not\ z\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}z$	$\{\boldsymbol{T}z, \boldsymbol{F}\{x, not\ c\}, \boldsymbol{F}\{not\ x, not\ y\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x, not\ z\}$	$\{\boldsymbol{T}y, \boldsymbol{F}\{x, not\ b\}, \boldsymbol{F}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x\}$	$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}a}$	$\{\boldsymbol{F}a, \boldsymbol{T}\{not\ x\}\}$	$\Delta_{\Pi_{11}}$	5
2	$\underline{\boldsymbol{T}b}$			30
	$\boldsymbol{T}\{not\ c\}$	$\{\boldsymbol{T}b, \boldsymbol{F}\{x\}, \boldsymbol{F}\{not\ c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ b\}$	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}c$	$\{\boldsymbol{T}\{not\ c\}, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{y, a, b\}}$	19
1	$\underline{\boldsymbol{T}y}$			
	$\boldsymbol{F}\{not\ y, not\ z\}$	$\{\boldsymbol{T}\{not\ y, not\ z\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{not\ x, not\ y\}$	$\{\boldsymbol{T}\{not\ x, not\ y\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	∇	
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{F}x, \boldsymbol{T}\{b, c\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{T}\{x, not\ b\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{x, not\ c\}$	$\{\boldsymbol{T}\{x, not\ c\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{x, not\ z\}$	$\{\boldsymbol{T}\{x, not\ z\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}z$	$\{\boldsymbol{T}z, \boldsymbol{F}\{x, not\ c\}, \boldsymbol{F}\{not\ x, not\ y\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{T}\{not\ x, not\ z\}$	$\{\boldsymbol{T}y, \boldsymbol{F}\{x, not\ b\}, \boldsymbol{F}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{T}\{not\ x\}$	$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	
	$\underline{\boldsymbol{T}a}$	$\{\boldsymbol{F}a, \boldsymbol{T}\{not\ x\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}b$			25
	$\boldsymbol{F}\{not\ c\}$	$\{\boldsymbol{F}b, \boldsymbol{T}\{not\ c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ b\}$	$\{\boldsymbol{F}\{not\ b\}, \boldsymbol{F}b\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}c}$	$\{\boldsymbol{F}\{not\ c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{y, a, c\}}$	19

Table A.4: First part of a computation of all answer sets with CDNL-ENUMERATION($\Pi_{11}, 0$).

dl	$\mathbf{A}$	δ	Info	Line
0	$\boldsymbol{F}y$			25
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{x, not\ b\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x, not\ z\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
1	$\underline{\boldsymbol{T}b}$			30
	$\boldsymbol{F}\{not\ b\}$	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	5
2	$\underline{\boldsymbol{T}c}$			30
	$\boldsymbol{F}\{not\ c\}$	$\{\boldsymbol{T}\{not\ c\}, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ c\}$	$\{\boldsymbol{T}\{x, not\ c\}, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{b, c\}$	$\{\boldsymbol{F}\{b, c\}, \boldsymbol{T}b, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{x\}$	$\{\boldsymbol{T}b, \boldsymbol{F}\{x\}, \boldsymbol{F}\{not\ c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{x, not\ z\}$	$\{\boldsymbol{T}c, \boldsymbol{F}\{x, not\ z\}, \boldsymbol{F}\{not\ b\}\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}a}$	$\{\boldsymbol{F}a, \boldsymbol{T}\{x\}\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}x}$	$\{\boldsymbol{F}x, \boldsymbol{T}\{b, c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x\}$	$\{\boldsymbol{T}\{not\ x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x, not\ y\}$	$\{\boldsymbol{T}\{not\ x, not\ y\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}z$	$\{\boldsymbol{T}\{x, not\ z\}, \boldsymbol{T}z\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ y, not\ z\}$	$\{\boldsymbol{F}\{not\ y, not\ z\}, \boldsymbol{F}y, \boldsymbol{F}z\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{x, a, b, c\}}$	19
1	$\boldsymbol{T}b$			
	$\boldsymbol{F}\{not\ b\}$	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}c$			25
	$\boldsymbol{T}\{not\ c\}$	$\{\boldsymbol{F}\{not\ c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{T}\{b, c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ z\}$	$\{\boldsymbol{F}c, \boldsymbol{T}\{x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
2	$\boldsymbol{F}a$			30
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{F}a, \boldsymbol{T}\{x\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x\}$	$\{\boldsymbol{F}a, \boldsymbol{T}\{not\ x\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}x$	$\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
		$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	6
		$\{\boldsymbol{F}a\}$	dl=**1**	9
0	$\boldsymbol{F}y$			
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{x, not\ b\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{not\ x, not\ z\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	
	$\underline{\boldsymbol{T}a}$	$\{\boldsymbol{F}a\}$	∇	5
1	$\underline{\boldsymbol{T}b}$			
	$\boldsymbol{F}\{not\ b\}$	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}c$			
	$\boldsymbol{T}\{not\ c\}$	$\{\boldsymbol{F}\{not\ c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{T}\{b, c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{x, not\ z\}$	$\{\boldsymbol{F}c, \boldsymbol{T}\{x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	
2	$\boldsymbol{F}x$			30
	$\boldsymbol{F}\{not\ y, not\ z\}$	$\{\boldsymbol{F}x, \boldsymbol{T}\{not\ y, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ c\}$	$\{\boldsymbol{T}\{x, not\ c\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x\}$	$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x, not\ y\}$	$\{\boldsymbol{F}\{not\ x, not\ y\}, \boldsymbol{F}x, \boldsymbol{F}y\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}z}$	$\{\boldsymbol{F}\{not\ x, not\ z\}, \boldsymbol{F}x, \boldsymbol{F}z\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{z, a, b\}}$	19

Table A.5: Second part of a computation of all answer sets with CDNL-ENUMERATION($\Pi_{11}, 0$).

dl	$\mathbf{A}$	δ	Info	Line
1	$\boldsymbol{T}b$			
	$\boldsymbol{F}\{not\ b\}$	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}c$			
	$\boldsymbol{T}\{not\ c\}$	$\{\boldsymbol{F}\{not\ c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{T}\{b, c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{x, not\ z\}$	$\{\boldsymbol{F}c, \boldsymbol{T}\{x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{T}x$			25
	$\boldsymbol{T}z$	$\{\boldsymbol{F}\{x, not\ z\}, \boldsymbol{T}x, \boldsymbol{F}z\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ y, not\ z\}$	$\{\boldsymbol{T}x, \boldsymbol{F}\{b, c\}, \boldsymbol{F}\{not\ y, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
		$\{\boldsymbol{T}\{not\ y, not\ z\}, \boldsymbol{T}z\}$	$\Delta_{\Pi_{11}}$	6
0	$\boldsymbol{F}y$			
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{x, not\ b\}\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{not\ x, not\ z\}$	$\{\boldsymbol{F}y, \boldsymbol{T}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	
	$\underline{\boldsymbol{T}a}$	$\{\boldsymbol{F}a\}$	∇	
	$\boldsymbol{F}b$			17
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{T}\{b, c\}, \boldsymbol{F}b\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ b\}$	$\{\boldsymbol{F}\{not\ b\}, \boldsymbol{F}b\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ c\}$	$\{\boldsymbol{F}b, \boldsymbol{T}\{not\ c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{F}b, \boldsymbol{T}\{x\}\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}c}$	$\{\boldsymbol{F}c, \boldsymbol{T}\{not\ b\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ c\}$	$\{\boldsymbol{T}\{x, not\ c\}, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}x$	$\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ y, not\ z\}$	$\{\boldsymbol{F}x, \boldsymbol{T}\{not\ y, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ z\}$	$\{\boldsymbol{T}\{x, not\ z\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x\}$	$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x, not\ y\}$	$\{\boldsymbol{F}\{not\ x, not\ y\}, \boldsymbol{F}x, \boldsymbol{F}y\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}z}$	$\{\boldsymbol{F}\{not\ x, not\ z\}, \boldsymbol{F}x, \boldsymbol{F}z\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{z, a, c\}}$	19

Table A.6: Third part of a computation of all answer sets with CDNL-ENUMERATION($\Pi_{11}, 0$).

dl	$\mathbf{A}$	δ	Info	Line
1	$\boldsymbol{T}y$			41
	$\boldsymbol{F}\{not\ y, not\ z\}$	$\{\boldsymbol{T}\{not\ y, not\ z\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x, not\ y\}$	$\{\boldsymbol{T}\{not\ x, not\ y\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	5
2	$\boldsymbol{T}a$			41
3	$\boldsymbol{T}x$			41
	$\boldsymbol{F}\{not\ x, not\ z\}$	$\{\boldsymbol{T}\{not\ x, not\ z\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{x, not\ b\}$	$\{\boldsymbol{T}y, \boldsymbol{F}\{x, not\ b\}, \boldsymbol{F}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}b$	$\{\boldsymbol{T}\{x, not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{F}b, \boldsymbol{T}\{x\}\}$	$\Delta_{\Pi_{11}}$	5
		$\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	6
		$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	dl=1	9
1	$\boldsymbol{T}y$			
	$\boldsymbol{F}\{not\ y, not\ z\}$	$\{\boldsymbol{T}\{not\ y, not\ z\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{not\ x, not\ y\}$	$\{\boldsymbol{T}\{not\ x, not\ y\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}x$	$\{\boldsymbol{T}x, \boldsymbol{T}y\}$	∇	5
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{F}x, \boldsymbol{T}\{b, c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{T}\{x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{T}\{x, not\ b\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ c\}$	$\{\boldsymbol{T}\{x, not\ c\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ z\}$	$\{\boldsymbol{T}\{x, not\ z\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}z$	$\{\boldsymbol{T}z, \boldsymbol{F}\{x, not\ c\}, \boldsymbol{F}\{not\ x, not\ y\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x, not\ z\}$	$\{\boldsymbol{T}y, \boldsymbol{F}\{x, not\ b\}, \boldsymbol{F}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x\}$	$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}a}$	$\{\boldsymbol{F}a, \boldsymbol{T}\{not\ x\}\}$	$\Delta_{\Pi_{11}}$	5
2	$\underline{\boldsymbol{T}b}$			41
	$\boldsymbol{T}\{not\ c\}$	$\{\boldsymbol{T}b, \boldsymbol{F}\{x\}, \boldsymbol{F}\{not\ c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ b\}$	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}c$	$\{\boldsymbol{T}\{not\ c\}, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{a, b\}}$	20
		$\{\boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}c\}$	$nogood(\mathbf{1})$	32
1	$\boldsymbol{T}b$		$decision(\mathbf{1})$	37
	$\boldsymbol{F}\{not\ b\}$	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{T}\{x, not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	5
2	$\boldsymbol{F}a$			41
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{F}a, \boldsymbol{T}\{x\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x\}$	$\{\boldsymbol{F}a, \boldsymbol{T}\{not\ x\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}x$	$\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
		$\{\boldsymbol{F}\{not\ x\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	6
		$\{\boldsymbol{F}a\}$	dl=**1**	9

Table A.7: First part of a computation of all projected answer sets with CDNL-PROJECTION($\Pi_{11}, \{a, b, c\}, 0$).

dl	**A**	δ	Info	Line
0	$\underline{\boldsymbol{T}a}$	$\{\boldsymbol{F}a\}$	∇	5
1	$\underline{\boldsymbol{T}b}$			
	$\boldsymbol{F}\{not\ b\}$	$\{\boldsymbol{T}\{not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{T}\{x, not\ b\}, \boldsymbol{T}b\}$	$\Delta_{\Pi_{11}}$	
	$\underline{\boldsymbol{T}c}$	$\{\boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}c\}$	$nogood(\mathbf{1})$	5
	$\boldsymbol{T}\{b, c\}$	$\{\boldsymbol{F}\{b, c\}, \boldsymbol{T}b, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ c\}$	$\{\boldsymbol{T}\{not\ c\}, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ c\}$	$\{\boldsymbol{T}\{x, not\ c\}, \boldsymbol{T}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{x\}$	$\{\boldsymbol{T}b, \boldsymbol{F}\{x\}, \boldsymbol{F}\{not\ c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{x, not\ z\}$	$\{\boldsymbol{T}c, \boldsymbol{F}\{x, not\ z\}, \boldsymbol{F}\{not\ b\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}x$	$\{\boldsymbol{F}x, \boldsymbol{T}\{b, c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x\}$	$\{\boldsymbol{T}\{not\ x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x, not\ y\}$	$\{\boldsymbol{T}\{not\ x, not\ y\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x, not\ z\}$	$\{\boldsymbol{T}\{not\ x, not\ z\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}z$	$\{\boldsymbol{T}\{x, not\ z\}, \boldsymbol{T}z\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}y$	$\{\boldsymbol{T}y, \boldsymbol{F}\{x, not\ b\}, \boldsymbol{F}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ y, not\ z\}$	$\{\boldsymbol{F}\{not\ y, not\ z\}, \boldsymbol{F}y, \boldsymbol{F}z\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{a, b, c\}}$	20
		~~$\{\boldsymbol{T}a, \boldsymbol{T}b, \boldsymbol{F}c\}$~~	$nogood(\mathbf{1})$	24
0	$\underline{\boldsymbol{T}a}$	$\{\boldsymbol{F}a\}$	∇	
	$\boldsymbol{F}b$			28
	$\boldsymbol{F}\{x\}$	$\{\boldsymbol{F}b, \boldsymbol{T}\{x\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ c\}$	$\{\boldsymbol{F}b, \boldsymbol{T}\{not\ c\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{b, c\}$	$\{\boldsymbol{T}\{b, c\}, \boldsymbol{F}b\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ b\}$	$\{\boldsymbol{F}\{not\ b\}, \boldsymbol{F}b\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}x$	$\{\boldsymbol{F}\{x\}, \boldsymbol{T}x\}$	$\Delta_{\Pi_{11}}$	5
	$\underline{\boldsymbol{T}c}$	$\{\boldsymbol{F}\{not\ c\}, \boldsymbol{F}c\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{T}\{not\ x\}$	$\{\boldsymbol{T}a, \boldsymbol{F}\{x\}, \boldsymbol{F}\{not\ x\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ y, not\ z\}$	$\{\boldsymbol{F}x, \boldsymbol{T}\{not\ y, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ b\}$	$\{\boldsymbol{T}\{x, not\ b\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ c\}$	$\{\boldsymbol{T}\{x, not\ c\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{x, not\ z\}$	$\{\boldsymbol{T}\{x, not\ z\}, \boldsymbol{F}x\}$	$\Delta_{\Pi_{11}}$	5
1	$\boldsymbol{T}y$			41
	$\boldsymbol{T}\{not\ x, not\ z\}$	$\{\boldsymbol{T}y, \boldsymbol{F}\{x, not\ b\}, \boldsymbol{F}\{not\ x, not\ z\}\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}\{not\ x, not\ y\}$	$\{\boldsymbol{T}\{not\ x, not\ y\}, \boldsymbol{T}y\}$	$\Delta_{\Pi_{11}}$	5
	$\boldsymbol{F}z$	$\{\boldsymbol{T}\{not\ x, not\ z\}, \boldsymbol{T}z\}$	$\Delta_{\Pi_{11}}$	5
			$\underline{\{a, c\}}$	20

Table A.8: Second part of a computation of all projected answer sets with CDNL-PROJECTION($\Pi_{11}, \{a, b, c\}, 0$).

Appendix B

Proofs

This appendix provides proofs of formal results in the thesis, structured by chapters.

B.1 Chapter 2

The results in Chapter 2 concentrate on properties of unfounded sets, and also compare our unfounded set notion with an adaption of the traditional concept [216], referred to by GRS-unfounded sets.

The first result parallels Theorem 5.4 in [216] and Theorem 4.6 in [159] by showing that a model of a normal program Π is an answer set of Π exactly if its intersection with each GRS-unfounded set is empty.

Proposition 2.1. *Let Π be a normal program and $\mathbf{A}$ a non-contradictory assignment.*

If $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is a model of Π, then we have that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is an answer set of Π iff $U \cap \mathbf{A}^{\boldsymbol{T}} = \emptyset$ holds for every GRS-unfounded set U of Π w.r.t. $\mathbf{A}$.

Proof. Assume that $X = \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is a model of Π. Then, X is a model of Π^X as well, so that $Cn(\Pi^X) \subseteq X$. Furthermore, one of the following cases applies:

($Cn(\Pi^X) = X$) We have that X is an answer set of Π. Furthermore, for any $U \subseteq atom(\Pi)$ such that $U \cap \mathbf{A}^{\boldsymbol{T}} \neq \emptyset$, it holds that $U \cap X \neq \emptyset$, so that $X \setminus U$ is not a model of Π^X. That is, there is a rule $r \in \Pi$ such that $head(r) \in U \cap X$, $body(r)^+ \subseteq X \setminus U$, and $body(r)^- \cap X = \emptyset$. From $head(r) \in U \cap X$, $body(r)^+ \subseteq X \setminus U$, and the prerequisite that $\mathbf{A}$ is non-contradictory, we conclude that $body(r) \in EB_\Pi(U)$,[1] $body(r)^+ \subseteq \mathbf{A}^{\boldsymbol{T}}$, and $body(r)^+ \cap \mathbf{A}^{\boldsymbol{F}} = \emptyset$. Since $body(r)^- \cap X = \emptyset$, we also have that $body(r)^- \cap \mathbf{A}^{\boldsymbol{T}} = \emptyset$. This shows that $EB_\Pi(U) \not\subseteq \{B \in body(\Pi) \mid (B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\}$, so that U is not a GRS-unfounded set of Π w.r.t. $\mathbf{A}$. In turn, every GRS-unfounded set U of Π w.r.t. $\mathbf{A}$ is such that $U \cap \mathbf{A}^{\boldsymbol{T}} = \emptyset$.

($Cn(\Pi^X) \subset X$) We have that X is not an answer set of Π. Furthermore, for $U = atom(\Pi) \setminus Cn(\Pi^X)$ and every rule $r \in \Pi$, it holds that $head(r) \notin U$, $body(r)^+ \not\subseteq Cn(\Pi^X)$, or $body(r)^- \cap X \neq \emptyset$. Since $body(r)^+ \not\subseteq Cn(\Pi^X)$ implies $body(r)^+ \cap U \neq \emptyset$, for every $B \in EB_\Pi(U)$, we have that $B^- \cap X = B^- \cap \mathbf{A}^{\boldsymbol{T}} \neq \emptyset$. This shows that $EB_\Pi(U) \subseteq \{B \in body(\Pi) \mid (B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\}$, so that U is a GRS-unfounded set of Π w.r.t. $\mathbf{A}$ such that $U \cap \mathbf{A}^{\boldsymbol{T}} = U \cap X \neq \emptyset$.

[1] Recall that $EB_\Pi(U) = \{body(r) \mid r \in \Pi, head(r) \in U, body(r)^+ \cap U = \emptyset\}$.

The above cases show that, if $X = \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is a model of Π, then X is an answer set of Π iff $U \cap \mathbf{A}^{\boldsymbol{T}} = \emptyset$ holds for every GRS-unfounded set U of Π w.r.t. $\mathbf{A}$. □

The next result links GRS-unfounded sets to unfounded sets by showing that, w.r.t. a body-saturated assignment, every GRS-unfounded set is an unfounded set as well.

Proposition 2.2. *Let Π be a normal program, $\mathbf{A}$ an assignment, and $U \subseteq atom(\Pi)$.*

If $\mathbf{A}$ is body-saturated for Π, then we have that U is an unfounded set of Π w.r.t. $\mathbf{A}$ if U is a GRS-unfounded set of Π w.r.t. $\mathbf{A}$.

Proof. Assume that $\mathbf{A}$ is body-saturated for Π. Then, for every $B \in body(\Pi)$, $(B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset$ implies $B \in \mathbf{A}^{\boldsymbol{F}}$. Hence, if U is a GRS-unfounded set of Π w.r.t. $\mathbf{A}$, then $EB_\Pi(U) \subseteq \{B \in body(\Pi) \mid (B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\}$ implies $EB_\Pi(U) \subseteq \mathbf{A}^{\boldsymbol{F}}$, so that U is an unfounded set of Π w.r.t. $\mathbf{A}$. □

Further considering the relationships between unfounded set concepts, unfounded sets and GRS-unfounded sets coincide w.r.t. body-synchronized assignments.

Proposition 2.3. *Let Π be a normal program, $\mathbf{A}$ an assignment, and $U \subseteq atom(\Pi)$.*

If $\mathbf{A}$ is body-synchronized for Π, then we have that U is an unfounded set of Π w.r.t. $\mathbf{A}$ iff U is a GRS-unfounded set of Π w.r.t. $\mathbf{A}$.

Proof. Assume that $\mathbf{A}$ is body-synchronized for Π. Then, $\mathbf{A}$ is body-saturated for Π according to Definition 2.3, and by Proposition 2.2, U is an unfounded set of Π w.r.t. $\mathbf{A}$ if U is a GRS-unfounded set of Π w.r.t. $\mathbf{A}$. It remains to show that the converse holds as well. Since $\mathbf{A}$ is body-synchronized for Π, for every $B \in body(\Pi)$, $B \in \mathbf{A}^{\boldsymbol{F}}$ implies $(B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset$. Hence, if U is an unfounded set of Π w.r.t. $\mathbf{A}$, then $EB_\Pi(U) \subseteq \mathbf{A}^{\boldsymbol{F}}$ implies $EB_\Pi(U) \subseteq \{B \in body(\Pi) \mid (B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\}$, so that U is a GRS-unfounded set of Π w.r.t. $\mathbf{A}$. □

In view of Proposition 2.3, Proposition 2.1 can be reformulated as follows.

Corollary 2.4. *Let Π be a normal program and $\mathbf{A}$ a non-contradictory assignment.*

If $\mathbf{A}$ is body-synchronized for Π and if $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is a model of Π, then we have that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is an answer set of Π iff $U \cap \mathbf{A}^{\boldsymbol{T}} = \emptyset$ holds for every unfounded set U of Π w.r.t. $\mathbf{A}$.

Proof. This result follows immediately from Proposition 2.1 and 2.3, since GRS-unfounded sets match unfounded sets w.r.t. a body-synchronized assignment $\mathbf{A}$. □

In what follows, we show crucial properties of unfounded sets. To begin with, false atoms of an unfounded set may be removed, while still maintaining unfoundedness, if the assignment at hand is body-saturated.

Proposition 2.5. *Let Π be a normal program, $\mathbf{A}$ an assignment, and U an unfounded set of Π w.r.t. $\mathbf{A}$.*

If $\mathbf{A}$ is body-saturated for Π, then we have that $U \setminus \mathbf{A}^{\boldsymbol{F}}$ is an unfounded set of Π w.r.t. $\mathbf{A}$.

Proof. Assume that $\mathbf{A}$ is body-saturated for Π. Then, for every $B \in EB_\Pi(U \setminus \mathbf{A}^{\boldsymbol{F}}) \setminus EB_\Pi(U)$, the fact that $B^+ \cap (U \cap \mathbf{A}^{\boldsymbol{F}}) \neq \emptyset$ implies $B \in \mathbf{A}^{\boldsymbol{F}}$. Along with $EB_\Pi(U) \subseteq \mathbf{A}^{\boldsymbol{F}}$, we conclude that $EB_\Pi(U \setminus \mathbf{A}^{\boldsymbol{F}}) \subseteq \mathbf{A}^{\boldsymbol{F}}$, so that $U \setminus \mathbf{A}^{\boldsymbol{F}}$ is an unfounded set of Π w.r.t. $\mathbf{A}$. □

Corollary 2.6. *Let* Π *be a normal program,* $\mathbf{A}$ *an assignment, and* U *an unfounded set of* Π *w.r.t.* $\mathbf{A}$.

If $\mathbf{A}$ *is body-saturated for* Π *and if* $U \not\subseteq \mathbf{A}^{\boldsymbol{F}}$*, then we have that* $U \setminus \mathbf{A}^{\boldsymbol{F}}$ *is a non-empty unfounded set of* Π *w.r.t.* $\mathbf{A}$.

Proof. This result follows immediately from Proposition 2.5 and the fact that $U \not\subseteq \mathbf{A}^{\boldsymbol{F}}$ implies $U \setminus \mathbf{A}^{\boldsymbol{F}} \neq \emptyset$. □

The following auxiliary result shows that, w.r.t. an atom-saturated assignment, any non-empty unfounded set of non-false atoms that is not a loop contains in turn a non-empty proper subset that is unfounded.

Lemma B.1. *Let* Π *be a normal program,* $\mathbf{A}$ *an assignment, and* $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ *a non-empty unfounded set of* Π *w.r.t.* $\mathbf{A}$.

If $\mathbf{A}$ *is atom-saturated for* Π *and if* $U \notin loop(\Pi)$*, then we have that there is some non-empty unfounded set* $U' \subset U$ *of* Π *w.r.t.* $\mathbf{A}$.

Proof. Assume that $\mathbf{A}$ is atom-saturated for Π and that $U \notin loop(\Pi)$. Then, for every $p \in U$, the prerequisite that $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ implies $body_\Pi(p) \not\subseteq \mathbf{A}^{\boldsymbol{F}}$, while $EB_\Pi(U) \subseteq \mathbf{A}^{\boldsymbol{F}}$ yields $body_\Pi(p) \cap EB_\Pi(U) \subseteq \mathbf{A}^{\boldsymbol{F}}$. That is, for every $p \in U$, there is some $B \in body_\Pi(p) \setminus \mathbf{A}^{\boldsymbol{F}}$, and $B^+ \cap U \neq \emptyset$ holds for each $B \in body_\Pi(p) \setminus \mathbf{A}^{\boldsymbol{F}}$. Hence, every atom of U has some successor belonging to U in $DG(\Pi) = (atom(\Pi), \{(head(r), p) \mid r \in \Pi, p \in body(r)^+\})$. However, since $U \notin loop(\Pi)$, we have that the subgraph of $DG(\Pi)$ induced by U is not strongly connected. Along with the fact that U is finite, we conclude that there is some strongly connected component of $(U, \{(head(r), p) \mid r \in \Pi, head(r) \in U, p \in body(r)^+ \cap U\})$ such that its vertices C are not reached from atoms in $U \setminus C$.[2] The latter means that $B^+ \cap C = \emptyset$ holds for every $B \in EB_\Pi(U \setminus C)$, so that $EB_\Pi(U \setminus C) \subseteq EB_\Pi(U)$. Since $\emptyset \subset C \subset U$ and $EB_\Pi(U) \subseteq \mathbf{A}^{\boldsymbol{F}}$, this shows that $U' = U \setminus C$ is a non-empty unfounded set of Π w.r.t. $\mathbf{A}$ such that $U' \subset U$. □

The previous lemma allows us to conclude that, w.r.t. an atom-saturated assignment, every non-empty unfounded set of non-false must contain an unfounded loop.

Proposition 2.7. *Let* Π *be a normal program,* $\mathbf{A}$ *an assignment, and* $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ *a non-empty unfounded set of* Π *w.r.t.* $\mathbf{A}$.

If $\mathbf{A}$ *is atom-saturated for* Π*, then we have that there is some unfounded set* $L \subseteq U$ *of* Π *w.r.t.* $\mathbf{A}$ *such that* $L \in loop(\Pi)$.

Proof. Assume that $\mathbf{A}$ is atom-saturated for Π. Then, since $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ is a non-empty unfounded set of Π w.r.t. $\mathbf{A}$, there is some non-empty unfounded set $L \subseteq U$ of Π w.r.t. $\mathbf{A}$ such that $\emptyset$ and L are all unfounded sets of Π w.r.t. $\mathbf{A}$ contained in L. For each such $L \subseteq U$, by Lemma B.1, we conclude that $L \in loop(\Pi)$. □

Corollary 2.8. *Let* Π *be a normal program,* $\mathbf{A}$ *an assignment, and* $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ *a non-empty unfounded set of* Π *w.r.t.* $\mathbf{A}$.

If $\mathbf{A}$ *is atom-saturated for* Π*, then we have that there is some non-empty unfounded set* $U' \subseteq U$ *of* Π *w.r.t.* $\mathbf{A}$ *such that all* $p \in U'$ *belong to the same non-trivial strongly connected component of* $DG(\Pi)$.

[2] Note that the "condensation" of $(U, \{(head(r), p) \mid r \in \Pi, head(r) \in U, p \in body(r)^+ \cap U\})$, obtained by contracting each strongly connected component to a single vertex, is a directed acyclic graph (cf. [199]).

Proof. This result follows immediately from Proposition 2.7, since all atoms of some $L \in \mathit{loop}(\Pi)$ belong to the same strongly connected component of $DG(\Pi)$, which must be non-trivial by the definition of a loop. □

Finally, we combine Corollary 2.6 and Proposition 2.7 to show that, w.r.t. an assignment that is both atom- and body-saturated, any unfounded set that includes non-false atoms must contain an unfounded loop of non-false atoms.

Theorem 2.9. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment.*

If $\mathbf{A}$ *is both atom- and body-saturated for* Π *and if there is some unfounded set* U *of* Π *w.r.t.* $\mathbf{A}$ *such that* $U \not\subseteq \mathbf{A}^{\boldsymbol{F}}$*, then we have that there is some unfounded set* $L \subseteq U \setminus \mathbf{A}^{\boldsymbol{F}}$ *of* Π *w.r.t.* $\mathbf{A}$ *such that* $L \in \mathit{loop}(\Pi)$*.*

Proof. Assume that $\mathbf{A}$ is both atom- and body-saturated for Π and that there is some unfounded set U of Π w.r.t. $\mathbf{A}$ such that $U \not\subseteq \mathbf{A}^{\boldsymbol{F}}$. Then, by Corollary 2.6, we have that $U \setminus \mathbf{A}^{\boldsymbol{F}}$ is a non-empty unfounded set of Π w.r.t. $\mathbf{A}$. Furthermore, by Proposition 2.7, there is some unfounded set $L \subseteq U \setminus \mathbf{A}^{\boldsymbol{F}}$ of Π w.r.t. $\mathbf{A}$ such that $L \in \mathit{loop}(\Pi)$. □

Corollary 2.10. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment.*

If $\mathbf{A}$ *is both atom- and body-saturated for* Π *and if there is some unfounded set* U *of* Π *w.r.t.* $\mathbf{A}$ *such that* $U \not\subseteq \mathbf{A}^{\boldsymbol{F}}$*, then we have that there is some non-empty unfounded set* $U' \subseteq U \setminus \mathbf{A}^{\boldsymbol{F}}$ *of* Π *w.r.t.* $\mathbf{A}$ *such that all* $p \in U'$ *belong to the same non-trivial strongly connected component of* $DG(\Pi)$*.*

Proof. This result follows immediately from Theorem 2.9, since all atoms of some $L \in \mathit{loop}(\Pi)$ belong to the same strongly connected component of $DG(\Pi)$, which must be non-trivial by the definition of a loop. □

We have thus proven all formal results presented in Chapter 2.

B.2 Chapter 3

We present proofs of results by sections. Proofs of Theorem 3.1 from Section 3.1 and Theorem 3.8 from Section 3.2 are postponed to Appendix B.2.2, where they can be derived as consequences of more general results.

B.2.1 Section 3.2

To begin with, we show Proposition 3.2 and 3.3 on correspondences between tableau rules and logic programming operators as well as *smodels*' propagation.

Proposition 3.2. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment.*

Then, we have that

1. $\mathbb{T}_\Pi(\mathbf{A}) = \big(D_{\{\mathit{FTA}\}}(\Pi, D_{\{\mathit{FTB}\}}(\Pi, \mathbf{A}))\big)^{\boldsymbol{T}}$;
2. $\mathbb{N}_\Pi(\mathbf{A}) = \big(D_{\{\mathit{FFA}\}}(\Pi, D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}))\big)^{\boldsymbol{F}}$;
3. $\mathbb{U}_\Pi(\mathbf{A}) = \Big(D_{\{\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}))\Big)^{\boldsymbol{F}}$.

Proof. We separately consider the items of the statement:

1. We have that $p \in \mathbb{T}_\Pi(\mathbf{A})$ iff $p = \mathit{head}(r)$ for some $r \in \Pi$ such that $\mathit{body}(r)^+ \subseteq \mathbf{A}^{\boldsymbol{T}}$ and $\mathit{body}(r)^- \subseteq \mathbf{A}^{\boldsymbol{F}}$ iff $p = \mathit{head}(r)$ for some $r \in \Pi$ such that $\boldsymbol{T}\mathit{body}(r) \in D_{\{\mathit{FTB}\}}(\Pi, \mathbf{A})$, so that $p \in \big(D_{\{\mathit{FTA}\}}(\Pi, D_{\{\mathit{FTB}\}}(\Pi, \mathbf{A}))\big)^{\boldsymbol{T}}$.

2. We have that $p \in \mathbb{N}_\Pi(\mathbf{A})$ iff $p \in \mathit{atom}(\Pi)$ such that $\mathit{head}(r) \neq p$ or $(\mathit{body}(r)^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (\mathit{body}(r)^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset$ for every $r \in \Pi$ iff $p \in \mathit{atom}(\Pi)$ such that $\boldsymbol{F}B \in D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A})$ for every $B \in \mathit{body}_\Pi(p)$, so that $p \in \big(D_{\{\mathit{FFA}\}}(\Pi, D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}))\big)^{\boldsymbol{F}}$.

3. We have that $p \in \mathbb{U}_\Pi(\mathbf{A})$ iff $p \in U$ for some $U \subseteq \mathit{atom}(\Pi)$ such that $(B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset$ for every $B \in EB_\Pi(U)$ iff $p \in U$ for some $U \subseteq \mathit{atom}(\Pi)$ such that $\boldsymbol{F}B \in D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A})$ for every $B \in EB_\Pi(U)$, so that $p \in \Big(D_{\{\mathit{WFN}[2^{\mathit{atom}(\Pi)}]\}}(\Pi, D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}))\Big)^{\boldsymbol{F}}$.

We have thus shown that all items of the statement hold. □

Proposition 3.3. *Let Π be a normal program and $\mathbf{A}$ an assignment.*
Then, we have that

1. $D_{\{\mathit{FI}\}}(\Pi, \mathbf{A}) = D_{\{\mathit{FTA}\}}(\Pi, D_{\{\mathit{FTB}\}}(\Pi, \mathbf{A}))$;
2. $D_{\{\mathit{ARC}\}}(\Pi, \mathbf{A}) = D_{\{\mathit{FFA}\}}(\Pi, D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}))$;
3. $D_{\{\mathit{CTH}\}}(\Pi, \mathbf{A}) = D_{\{\mathit{BTB}\}}(\Pi, D_{\{\mathit{BTA}\}}(\Pi, D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}) \cup \{\boldsymbol{T}p \mid p \in \mathbf{A}^{\boldsymbol{T}} \cap \mathit{atom}(\Pi)\}))$;
4. $D_{\{\mathit{CFH}\}}(\Pi, \mathbf{A}) = D_{\{\mathit{BFB}\}}(\Pi, D_{\{\mathit{BFA}\}}(\Pi, \mathbf{A}) \cup \{\boldsymbol{T}p \mid p \in \mathbf{A}^{\boldsymbol{T}} \cap \mathit{atom}(\Pi)\} \cup \{\boldsymbol{F}p \mid p \in \mathbf{A}^{\boldsymbol{F}} \cap \mathit{atom}(\Pi)\})$;
5. $D_{\{\mathit{AM}\}}(\Pi, \mathbf{A}) = D_{\{\mathit{WFN}[2^{\mathit{atom}(\Pi)}]\}}(\Pi, D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}))$.

Proof. We separately consider the items of the statement:

1. We have that $\boldsymbol{T}p \in D_{\{\mathit{FI}\}}(\Pi, \mathbf{A})$ iff $p = \mathit{head}(r)$ for some $r \in \Pi$ such that $\mathit{body}(r)^+ \subseteq \mathbf{A}^{\boldsymbol{T}}$ and $\mathit{body}(r)^- \subseteq \mathbf{A}^{\boldsymbol{F}}$ iff $p = \mathit{head}(r)$ for some $r \in \Pi$ such that $\boldsymbol{T}\mathit{body}(r) \in D_{\{\mathit{FTB}\}}(\Pi, \mathbf{A})$, so that $\boldsymbol{T}p \in D_{\{\mathit{FTA}\}}(\Pi, D_{\{\mathit{FTB}\}}(\Pi, \mathbf{A}))$.

2. We have that $\boldsymbol{F}p \in D_{\{\mathit{ARC}\}}(\Pi, \mathbf{A})$ iff $p \in \mathit{atom}(\Pi)$ such that $(B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset$ for every $B \in \mathit{body}_\Pi(p)$ iff $p \in \mathit{atom}(\Pi)$ such that $\boldsymbol{F}B \in D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A})$ for every $B \in \mathit{body}_\Pi(p)$, so that $\boldsymbol{F}p \in D_{\{\mathit{FFA}\}}(\Pi, D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}))$.

3. We have that $\boldsymbol{t}l \in D_{\{\mathit{CTH}\}}(\Pi, \mathbf{A})$ iff $p \in \mathbf{A}^{\boldsymbol{T}} \cap \mathit{atom}(\Pi)$ and $l \in \mathit{body}(r)$ for some $r \in \Pi$ such that $(B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset$ for every $B \in \mathit{body}_\Pi(p) \setminus \{\mathit{body}(r)\}$ iff $p \in \mathbf{A}^{\boldsymbol{T}} \cap \mathit{atom}(\Pi)$ and $l \in \mathit{body}(r)$ for some $r \in \Pi$ such that $\{\boldsymbol{F}B \mid B \in \mathit{body}_\Pi(p) \setminus \{\mathit{body}(r)\}\} \subseteq D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A})$, so that $\boldsymbol{T}\mathit{body}(r) \in D_{\{\mathit{BTA}\}}(\Pi, D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}) \cup \{\boldsymbol{T}p \mid p \in \mathbf{A}^{\boldsymbol{T}} \cap \mathit{atom}(\Pi)\})$ and $\{\boldsymbol{t}l \mid l \in \mathit{body}(r)\} \subseteq D_{\{\mathit{BTB}\}}(\Pi, D_{\{\mathit{BTA}\}}(\Pi, D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}) \cup \{\boldsymbol{T}p \mid p \in \mathbf{A}^{\boldsymbol{T}} \cap \mathit{atom}(\Pi)\}))$.

4. We have that $\boldsymbol{f}l \in D_{\{\mathit{CFH}\}}(\Pi, \mathbf{A})$ iff $l \in \mathit{body}(r)$ for some $r \in \Pi$ such that $\boldsymbol{F}\mathit{head}(r) \in \mathbf{A}$ and $\boldsymbol{t}l' \in \mathbf{A}$ for every $l' \in \mathit{body}(r) \setminus \{l\}$ iff $\boldsymbol{F}\mathit{body}(r) \in D_{\{\mathit{BFA}\}}(\Pi, \mathbf{A})$ and $\{\boldsymbol{t}l' \mid l' \in \mathit{body}(r) \setminus \{l\}\} \subseteq \{\boldsymbol{T}p \mid p \in \mathbf{A}^{\boldsymbol{T}} \cap \mathit{atom}(\Pi)\} \cup$

$\{\boldsymbol{F}p \mid p \in \mathbf{A}^{\boldsymbol{F}} \cap atom(\Pi)\}$ for some $r \in \Pi$ and $l \in body(r)$, so that $\boldsymbol{f}l \in D_{\{BFB\}}(\Pi, D_{\{BFA\}}(\Pi, \mathbf{A}) \cup \{\boldsymbol{T}p \mid p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)\} \cup \{\boldsymbol{F}p \mid p \in \mathbf{A}^{\boldsymbol{F}} \cap atom(\Pi)\})$.

5. We have that $\boldsymbol{F}p \in D_{\{AM\}}(\Pi, \mathbf{A})$ iff $p \in U$ for some $U \subseteq atom(\Pi)$ such that $(B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset$ for every $B \in EB_\Pi(U)$ iff $p \in U$ for some $U \subseteq atom(\Pi)$ such that $\boldsymbol{F}B \in D_{\{FFB\}}(\Pi, \mathbf{A})$ for every $B \in EB_\Pi(U)$, so that $\boldsymbol{F}p \in D_{\{WFN[2^{atom(\Pi)}]\}}(\Pi, D_{\{FFB\}}(\Pi, \mathbf{A}))$.

We have thus shown that all items of the statement hold. □

In view of Proposition 3.3, we derive the following relationship between tableau calculi using the deterministic tableau rules in Figure 3.1 or 3.3, respectively.

Corollary 3.4. *Let Π be a normal program and $\mathbf{A}$ an assignment.*
Then, we have that $D^*_{\{FI,ARC,CTH,CFH,AM\}}(\Pi, \mathbf{A}) \subseteq D^*_{\mathcal{T}_{smodels}}(\Pi, \mathbf{A})$.

Proof. This result follows immediately from Proposition 3.3, since any entry deducible by some of the tableau rules in $\{FI, ARC, CTH, CFH, AM\}$ can likewise be deduced by iterated applications of the tableau rules *(a)–(h)* and $WFN[2^{atom(\Pi)}]$ in Figure 3.1, which are the deterministic tableau rules contained in $\mathcal{T}_{smodels}$. □

Next, we show the one-to-one correspondence between models of $Comp(\Pi)$ and non-contradictory complete branches in tableaux of $\mathcal{T}_{comp}$, stated in Theorem 3.5. To this end, we provide Lemma B.2, derived from Proposition 4.1 in Section 4.1. Lemma B.2 is applied only in the proof of Theorem 3.5, which itself is not used to prove any of the other results in this thesis.

Lemma B.2. *Let Π be a normal program and $X \subseteq atom(\Pi) \cup body(\Pi)$.*
Then, we have that $(X \cap atom(\Pi)) \cup \{p_B \mid B \in X \cap body(\Pi)\}$ is a model of $Comp(\Pi)$ iff $D_{\{(a)–(h)\}}(\Pi, \mathbf{A}) \subseteq \{\boldsymbol{T}v \mid v \in X\} \cup \{\boldsymbol{F}v \mid v \in (atom(\Pi) \cup body(\Pi)) \setminus X\}$ for every assignment $\mathbf{A} \subseteq \{\boldsymbol{T}v \mid v \in X\} \cup \{\boldsymbol{F}v \mid v \in (atom(\Pi) \cup body(\Pi)) \setminus X\}$.

Proof. By Proposition 4.1, we have that $(X \cap atom(\Pi)) \cup \{p_B \mid B \in X \cap body(\Pi)\}$ is a model of $Comp(\Pi)$ iff $\mathbf{A} = \{\boldsymbol{T}v \mid v \in X\} \cup \{\boldsymbol{F}v \mid v \in (atom(\Pi) \cup body(\Pi)) \setminus X\}$ is a solution for Δ_Π. We make use of this relationship in the following consideration of the implications of the statement:

(⇒) Assume that $\mathbf{A}' \subseteq \mathbf{A}$ but $D_{\{(a)–(h)\}}(\Pi, \mathbf{A}') \not\subseteq \mathbf{A}$. Then, some of the following cases applies:

($D_{\{FTB,BFB\}}(\Pi, \mathbf{A}') \not\subseteq \mathbf{A}$) We have that $\{\boldsymbol{F}\{l_1, \ldots, l_n\}, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\} \subseteq \mathbf{A}$ for some $\{l_1, \ldots, l_n\} \in body(\Pi)$. Since $\{\boldsymbol{F}\{l_1, \ldots, l_n\}, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\} \in \Delta_\Pi$, this shows that $\mathbf{A}$ is not a solution for Δ_Π.

($D_{\{FFB,BTB\}}(\Pi, \mathbf{A}') \not\subseteq \mathbf{A}$) We have that $\{\boldsymbol{T}\{l_1, \ldots, l_i, \ldots, l_n\}, \boldsymbol{f}l_i\} \subseteq \mathbf{A}$ for some $\{l_1, \ldots, l_i, \ldots, l_n\} \in body(\Pi)$. Since $\{\boldsymbol{T}\{l_1, \ldots, l_i, \ldots, l_n\}, \boldsymbol{f}l_i\} \in \Delta_\Pi$, this shows that $\mathbf{A}$ is not a solution for Δ_Π.

($D_{\{FTA,BFA\}}(\Pi, \mathbf{A}') \not\subseteq \mathbf{A}$) We have that $\{\boldsymbol{F}p, \boldsymbol{T}B\} \subseteq \mathbf{A}$ for some $p \in atom(\Pi)$ and $B \in body_\Pi(p)$. Since $\{\boldsymbol{F}p, \boldsymbol{T}B\} \in \Delta_\Pi$, this shows that $\mathbf{A}$ is not a solution for Δ_Π.

($D_{\{FFA,BTA\}}(\Pi, \mathbf{A}') \not\subseteq \mathbf{A}$) We have that $\{\boldsymbol{T}p, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \subseteq \mathbf{A}$ for some $p \in atom(\Pi)$ and $body_\Pi(p) = \{B_1, \ldots, B_k\}$. Since $\{\boldsymbol{T}p, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \in \Delta_\Pi$, this shows that $\mathbf{A}$ is not a solution for Δ_Π.

In each of the above cases, $\mathbf{A}$ is not a solution for Δ_Π, so that $(X \cap atom(\Pi)) \cup \{p_B \mid B \in X \cap body(\Pi)\}$ is not a model of $Comp(\Pi)$.

($\Leftarrow$) Assume that $\mathbf{A}$ is not a solution for Δ_Π. Then, some of the following cases applies:

($\{\boldsymbol{F}\{l_1, \ldots, l_n\}, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\} \subseteq \mathbf{A}$ for some $\{l_1, \ldots, l_n\} \in body(\Pi)$) We have that $\boldsymbol{T}\{l_1, \ldots, l_n\} \in D_{\{FTB\}}(\Pi, \mathbf{A})$, so that $D_{\{(a)\text{–}(h)\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$.

($\{\boldsymbol{T}\{l_1, \ldots, l_i, \ldots, l_n\}, \boldsymbol{f}l_i\} \subseteq \mathbf{A}$ for some $\{l_1, \ldots, l_i, \ldots, l_n\} \in body(\Pi)$) We have that $\boldsymbol{F}\{l_1, \ldots, l_i, \ldots, l_n\} \in D_{\{FFB\}}(\Pi, \mathbf{A})$, so that $D_{\{(a)\text{–}(h)\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$.

($\{\boldsymbol{F}p, \boldsymbol{T}B\} \subseteq \mathbf{A}$ for some $p \in atom(\Pi)$ and $B \in body_\Pi(p)$) We have that $\boldsymbol{T}p \in D_{\{FTA\}}(\Pi, \mathbf{A})$, so that $D_{\{(a)\text{–}(h)\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$.

($\{\boldsymbol{T}p, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \subseteq \mathbf{A}$ for some $p \in atom(\Pi)$ and $body_\Pi(p) = \{B_1, \ldots, B_k\}$) We have that $\boldsymbol{F}p \in D_{\{FFA\}}(\Pi, \mathbf{A})$, so that $D_{\{(a)\text{–}(h)\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$.

In each of the above cases, we have that $D_{\{(a)\text{–}(h)\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$, where $\mathbf{A} \subseteq \mathbf{A}$ trivially holds. □

Theorem 3.5. *Let Π be a normal program.*
Then, we have that the following holds for tableau calculus $\mathcal{T}_{comp}$:

1. *Every incomplete tableau for Π and $\emptyset$ can be extended to a complete tableau for Π and $\emptyset$.*
2. *$Comp(\Pi)$ has a model X iff every complete tableau for Π and $\emptyset$ has a unique non-contradictory branch $(\Pi, \mathbf{A})$ such that $(\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)) \cup \{p_B \mid B \in \mathbf{A}^{\boldsymbol{T}} \cap body(\Pi)\} = X$.*
3. *$Comp(\Pi)$ has no model iff every complete tableau for Π and $\emptyset$ is a refutation.*

Proof. We separately consider the items of the statement:

1. By applying *Cut*$[atom(\Pi) \cup body(\Pi)]$, an incomplete branch in a tableau for Π and $\emptyset$ can be extended to a subtableau such that, for every branch $(\Pi, \mathbf{A})$ in it, we have that $atom(\Pi) \cup body(\Pi) \subseteq \mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}}$. Furthermore, if $(\Pi, \mathbf{A})$ is not complete, then $D_{\{(a)\text{–}(h)\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$, so that the application of some of the tableau rules *(a)–(h)* in $\mathcal{T}_{comp}$ yields a contradictory and thus complete branch.

2. We separately show the implications of the second item:

   ($\Rightarrow$) Assume that $X \subseteq atom(\Pi) \cup \{p_B \mid B \in body(\Pi)\}$ is a model of $Comp(\Pi)$, and consider the following assignment:

   $$\begin{aligned}\mathbf{A} \;=\; & \{\boldsymbol{T}p \mid p \in X \cap atom(\Pi)\} \cup \{\boldsymbol{F}p \mid p \in atom(\Pi) \setminus X\} \\ \cup\; & \{\boldsymbol{T}B \mid B \in body(\Pi), p_B \in X\} \cup \{\boldsymbol{F}B \mid B \in body(\Pi), p_B \notin X\}\end{aligned}$$

   Then, by Lemma B.2, $D_{\{(a)\text{–}(h)\}}(\Pi, \mathbf{A}') \subseteq \mathbf{A}$ for every assignment $\mathbf{A}' \subseteq \mathbf{A}$. Since either $\mathbf{A}' \cup \{\boldsymbol{T}v\} \subseteq \mathbf{A}$ or $\mathbf{A}' \cup \{\boldsymbol{F}v\} \subseteq \mathbf{A}$ for any application of

Cut$[atom(\Pi) \cup body(\Pi)]$ on a branch $(\Pi, \mathbf{A}')$ such that $\mathbf{A}' \subseteq \mathbf{A}$, we have that the assignment in exactly one of the resulting branches is contained in $\mathbf{A}$. Along with $\emptyset \subseteq \mathbf{A}$, it follows that every complete tableau for Π and $\emptyset$ has a unique non-contradictory branch $(\Pi, \mathbf{A})$ such that $(\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)) \cup \{p_B \mid B \in \mathbf{A}^{\boldsymbol{T}} \cap body(\Pi)\} = X$.

($\Leftarrow$) Assume that $(\Pi, \mathbf{A})$ is a non-contradictory complete branch, that is, $\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}} = atom(\Pi) \cup body(\Pi)$ and $D_{\{(a)\text{–}(h)\}}(\Pi, \mathbf{A}) \subseteq \mathbf{A}$. Then, by Lemma B.2 (along with the fact that $D_{\{(a)\text{–}(h)\}}(\Pi, \mathbf{A}') \subseteq D_{\{(a)\text{–}(h)\}}(\Pi, \mathbf{A})$ for every $\mathbf{A}' \subseteq \mathbf{A}$), we have that $X = (\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)) \cup \{p_B \mid B \in \mathbf{A}^{\boldsymbol{T}} \cap body(\Pi)\}$ is a model of $Comp(\Pi)$.

3. From the second item, if $Comp(\Pi)$ has a model, then every complete tableau for Π and $\emptyset$ has a non-contradictory branch; by the first item, there is some complete tableau for Π and $\emptyset$, so that some complete tableau for Π and $\emptyset$ is not a refutation. Conversely, if some complete tableau for Π and $\emptyset$ is not a refutation, it has a non-contradictory branch $(\Pi, \mathbf{A})$, and $(\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)) \cup \{p_B \mid B \in \mathbf{A}^{\boldsymbol{T}} \cap body(\Pi)\}$ is a model of $Comp(\Pi)$, as shown in the proof of the second item.

We have thus shown that all items of the statement hold. □

For proving Proposition 3.6, stating that tableau rule *WFN*$[2^{atom(\Pi)}]$ is as powerful as the iterated application of more restrictive tableau rules *FFA* and *WFN*$[loop(\Pi)]$ (along with *FFB*), we make use of Lemma 3.7 and B.3, the latter relying on Theorem 2.9.

Lemma 3.7. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment.*
Then, we have that

1. $\mathbf{A}$ *is body-saturated for* Π *iff* $D_{\{FFB\}}(\Pi, \mathbf{A}) \subseteq \mathbf{A}$*;*

2. $\mathbf{A}$ *is atom-saturated for* Π *iff* $D_{\{FFA\}}(\Pi, \mathbf{A}) \subseteq \mathbf{A}$.

Proof. We separately consider the items of the statement:

1. We have that $\boldsymbol{F}B \in D_{\{FFB\}}(\Pi, \mathbf{A})$ iff $B \in body(\Pi)$ such that $(B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset$. Hence, $D_{\{FFB\}}(\Pi, \mathbf{A}) \subseteq \mathbf{A}$ holds iff $\{B \in body(\Pi) \mid (B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\} \subseteq \mathbf{A}^{\boldsymbol{F}}$ iff $\mathbf{A}$ is body-saturated for Π according to Definition 2.3.

2. We have that $\boldsymbol{F}p \in D_{\{FFA\}}(\Pi, \mathbf{A})$ iff $p \in atom(\Pi)$ such that $body_\Pi(p) \subseteq \mathbf{A}^{\boldsymbol{F}}$. Hence, $D_{\{FFA\}}(\Pi, \mathbf{A}) \subseteq \mathbf{A}$ holds iff $\{p \in atom(\Pi) \mid body_\Pi(p) \subseteq \mathbf{A}^{\boldsymbol{F}}\} \subseteq \mathbf{A}^{\boldsymbol{F}}$ iff $\mathbf{A}$ is atom-saturated for Π according to Definition 2.4.

We have thus shown that the items of the statement hold. □

Lemma B.3. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment.*
Then, we have that $D_{\{FFB,WFN[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) \subseteq \mathbf{A}$ *iff* $D_{\{FFB,FFA,WFN[loop(\Pi)]\}}(\Pi, \mathbf{A}) \subseteq \mathbf{A}$.

Proof. We separately show the implications of the statement:

($\Rightarrow$) Assume that $D_{\{\mathit{FFB},\mathit{FFA},\mathit{WFN}[\mathit{loop}(\Pi)]\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$. Then, $D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$ or $D_{\{\mathit{FFA},\mathit{WFN}[\mathit{loop}(\Pi)]\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$. If $D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$, it is clear that $D_{\{\mathit{FFB},\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$. Otherwise, if $D_{\{\mathit{FFA},\mathit{WFN}[\mathit{loop}(\Pi)]\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$, there is some $p \in atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ such that $EB_\Pi(\{p\}) \subseteq \mathit{body}_\Pi(p) \subseteq \mathbf{A}^{\boldsymbol{F}}$ or $p \in L$ for an $L \in \mathit{loop}(\Pi)$ satisfying $EB_\Pi(L) \subseteq \mathbf{A}^{\boldsymbol{F}}$. Given that $\{\{p\} \mid p \in atom(\Pi)\} \cup \mathit{loop}(\Pi) \subseteq 2^{atom(\Pi)}$, we conclude that there is some $p \in atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ such that $\boldsymbol{F}p \in D_{\{\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A})$, so that $D_{\{\mathit{FFB},\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$.

($\Leftarrow$) Assume that $D_{\{\mathit{FFB},\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$. Then, $D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$ or $D_{\{\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$. If $D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$, it is clear that $D_{\{\mathit{FFB},\mathit{FFA},\mathit{WFN}[\mathit{loop}(\Pi)]\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$. Otherwise, if $D_{\{\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$, there is some $U \subseteq atom(\Pi)$ such that $U \not\subseteq \mathbf{A}^{\boldsymbol{F}}$ and $EB_\Pi(U) \subseteq \mathbf{A}^{\boldsymbol{F}}$, that is, U is an unfounded set of Π w.r.t. $\mathbf{A}$ such that $U \not\subseteq \mathbf{A}^{\boldsymbol{F}}$. Since $D_{\{\mathit{FFB},\mathit{FFA},\mathit{WFN}[\mathit{loop}(\Pi)]\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$ if $D_{\{\mathit{FFB},\mathit{FFA}\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$, assume that $D_{\{\mathit{FFB},\mathit{FFA}\}}(\Pi, \mathbf{A}) \subseteq \mathbf{A}$. Then, by Lemma 3.7, we have that $\mathbf{A}$ is both atom- and body-saturated for Π. Along with Theorem 2.9, we conclude that there is some $L \in \mathit{loop}(\Pi)$ such that $L \not\subseteq \mathbf{A}^{\boldsymbol{F}}$ and $EB_\Pi(L) \subseteq \mathbf{A}^{\boldsymbol{F}}$, so that $D_{\{\mathit{WFN}[\mathit{loop}(\Pi)]\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$ and $D_{\{\mathit{FFB},\mathit{FFA},\mathit{WFN}[\mathit{loop}(\Pi)]\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$. □

Proposition 3.6. *Let Π be a normal program and $\mathbf{A}$ an assignment.*

Then, we have that $D^*_{\{\mathit{FFB},\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) = D^*_{\{\mathit{FFB},\mathit{FFA},\mathit{WFN}[\mathit{loop}(\Pi)]\}}(\Pi, \mathbf{A})$.

Proof. By Lemma B.3, we have that $D^*_{\{\mathit{FFB},\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A})$ is closed under $\{\mathit{FFB}, \mathit{FFA}, \mathit{WFN}[\mathit{loop}(\Pi)]\}$ and that $D^*_{\{\mathit{FFB},\mathit{FFA},\mathit{WFN}[\mathit{loop}(\Pi)]\}}(\Pi, \mathbf{A})$ is closed under $\{\mathit{FFB}, \mathit{WFN}[2^{atom(\Pi)}]\}$. Along with the fact that $D^*_{\{\mathit{FFB},\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A})$ and $D^*_{\{\mathit{FFB},\mathit{FFA},\mathit{WFN}[\mathit{loop}(\Pi)]\}}(\Pi, \mathbf{A})$ are the unique smallest branches that extend $(\Pi, \mathbf{A})$ and are closed under $\{\mathit{FFB}, \mathit{WFN}[2^{atom(\Pi)}]\}$ or $\{\mathit{FFB}, \mathit{FFA}, \mathit{WFN}[\mathit{loop}(\Pi)]\}$, respectively, we conclude that $D^*_{\{\mathit{FFB},\mathit{FFA},\mathit{WFN}[\mathit{loop}(\Pi)]\}}(\Pi, \mathbf{A}) \subseteq D^*_{\{\mathit{FFB},\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A})$ and that $D^*_{\{\mathit{FFB},\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) \subseteq D^*_{\{\mathit{FFB},\mathit{FFA},\mathit{WFN}[\mathit{loop}(\Pi)]\}}(\Pi, \mathbf{A})$. □

We have thus proven the formal results presented in Section 3.2, except for Theorem 3.8, whose proof is provided at the end of Appendix B.2.2.

B.2.2 Section 3.3

For proving the soundness and completeness of our generic tableau method relative to the language constructs considered in Section 3.3, we first provide some lemmas. The correspondences between generic tableau rules and the basic ones for normal programs, as introduced in Section 3.1, then allow us to derive Theorem 3.1 and 3.8 as consequences of more general results.

Lemmas on Soundness

The first two lemmas provide properties of non-contradictory complete branches that hold in view of the generic tableau rules in Figure 3.4 on Page 34.

Lemma B.4. *Let Π be a disjunctive program and $\mathcal{T}$ a tableau calculus such that $\{I{\uparrow}, I{\downarrow}\} \cap \mathcal{T} \neq \emptyset$.*

Then, for every non-contradictory complete branch $(\Pi, \mathbf{A})$ and every $(\alpha \leftarrow \beta) \in \Pi$, we have that $\boldsymbol{t}\beta \notin \mathbf{A}$ or $\boldsymbol{f}\alpha \notin \mathbf{A}$.

Proof. Consider any $(\alpha \leftarrow \beta) \in \Pi$ and any branch $(\Pi, \mathbf{A})$ such that $\boldsymbol{t}\beta \in \mathbf{A}$ and $\boldsymbol{f}\alpha \in \mathbf{A}$. Then, we have that $\boldsymbol{t}\alpha \in D_{\{I\uparrow\}}(\Pi, \mathbf{A})$ and $\boldsymbol{f}\beta \in D_{\{I\downarrow\}}(\Pi, \mathbf{A})$. Since $\{I\uparrow, I\downarrow\} \cap \mathcal{T} \neq \emptyset$, this shows that $(\Pi, \mathbf{A})$ cannot be (extended to) a non-contradictory complete branch. □

Lemma B.5. *Let Π be a disjunctive program and $\mathcal{T}$ a tableau calculus such that $U\uparrow \in \mathcal{T}$.*

Then, for every non-contradictory complete branch $(\Pi, \mathbf{A})$ and every $S \subseteq atom(\Pi)$, we have that $sup_{\mathbf{A}}(\Pi, S, S) \neq \emptyset$ or $\mathbf{A}^{\boldsymbol{T}} \cap S = \emptyset$.

Proof. Consider any $S \subseteq atom(\Pi)$ and any branch $(\Pi, \mathbf{A})$ such that $sup_{\mathbf{A}}(\Pi, S, S) = \emptyset$ and $\mathbf{A}^{\boldsymbol{T}} \cap S \neq \emptyset$. Then, there is some $p \in \mathbf{A}^{\boldsymbol{T}} \cap S$ such that $\boldsymbol{F}p \in D_{\{U\uparrow\}}(\Pi, \mathbf{A})$. Since $U\uparrow \in \mathcal{T}$, this shows that $(\Pi, \mathbf{A})$ cannot be (extended to) a non-contradictory complete branch. □

For non-contradictory complete branches $(\Pi, \mathbf{A})$, the next lemmas show that the truth value of a variable $v \in atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \cup disj(\Pi)$ matches the valuation of $\tau[v]$ w.r.t. $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$, provided the inclusion of appropriate tableau rules, presented in Figure 3.5, 3.7, and 3.8 on Page 37, 41, and 44, respectively, in a calculus.

Lemma B.6. *Let Π be a disjunctive program and $\mathbf{A}$ a total assignment.*

Then, for every $p \in atom(\Pi)$, we have that

1. $\boldsymbol{t}p \in \mathbf{A}$ *iff* $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[p]$;
2. $\boldsymbol{t}not\ p \in \mathbf{A}$ *iff* $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[not\ p]$;
3. $\boldsymbol{f}p \in \mathbf{A}$ *iff* $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models \tau[p]$;
4. $\boldsymbol{f}not\ p \in \mathbf{A}$ *iff* $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models \tau[not\ p]$.

Proof. We have that $\tau[p] = p$ and $\tau[not\ p] = \neg\tau[p] = \neg p$, and the following holds:

1. $\boldsymbol{t}p \in \mathbf{A}$ iff $\boldsymbol{T}p \in \mathbf{A}$ iff $p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models p$;
2. $\boldsymbol{t}not\ p \in \mathbf{A}$ iff $\boldsymbol{F}p \in \mathbf{A}$ iff $p \notin \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \neg p$;
3. $\boldsymbol{f}p \in \mathbf{A}$ iff $\boldsymbol{F}p \in \mathbf{A}$ iff $p \notin \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models p$;
4. $\boldsymbol{f}not\ p \in \mathbf{A}$ iff $\boldsymbol{T}p \in \mathbf{A}$ iff $p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models \neg p$.

We have thus shown that all items of the statement hold. □

Lemma B.7. *Let Π be a disjunctive program and $\mathcal{T}$ a tableau calculus such that $\{TLU\uparrow, FL\uparrow, FU\uparrow\} \subseteq \mathcal{T}$.*

Then, for every non-contradictory complete branch $(\Pi, \mathbf{A})$ and every $v \in card(\Pi)$, we have that $\boldsymbol{T}v \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[v]$.

Proof. Consider any $v = j\{l_1, \ldots, l_n\}k \in card(\Pi)$ and any non-contradictory complete branch $(\Pi, \mathbf{A})$. For every $l \in \{l_1, \ldots, l_n\}$, we have that $l \in atom(\Pi)$ or $l = not\ p$ for some $p \in atom(\Pi)$. By Lemma B.6, $\boldsymbol{t}l \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[l]$, and $\boldsymbol{f}l \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models \tau[l]$. We further consider the cases that $\boldsymbol{T}v \in \mathbf{A}$ and $\boldsymbol{F}v \in \mathbf{A}$, respectively:

1. If $\boldsymbol{T}v \in \mathbf{A}$, then $\boldsymbol{F}v \notin D_{\{FL\uparrow,FU\uparrow\}}(\Pi, \mathbf{A})$. That is, $|\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{f}l \in \mathbf{A}\}| \leq n - j$ and $|\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\}| \leq k$. In view of $|\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\}| + |\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{f}l \in \mathbf{A}\}| = n$, $|\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{f}l \in \mathbf{A}\}| \leq n - j$ yields $j \leq |\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\}|$. We have thus shown that $j \leq |\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\}| \leq k$. Hence, for any $L \subseteq \{l_1, \ldots, l_n\}$ such that $|L| < j$, it holds that $\{l \in \{l_1, \ldots, l_n\} \setminus L \mid \boldsymbol{t}l \in \mathbf{A}\} \neq \emptyset$, so that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L} \tau[l])$. Moreover, for any $L \subseteq \{l_1, \ldots, l_n\}$ such that $k < |L|$, it holds that $\{l \in L \mid \boldsymbol{f}l \in \mathbf{A}\} \neq \emptyset$, so that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models (\bigwedge_{l \in L} \tau[l])$. Combining the cases for $|L| < j$ and $k < |L|$ yields that

$$\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \bigwedge_{\substack{L \subseteq \{l_1,\ldots,l_n\}, \\ |L|<j \text{ or } k<|L|}} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L} \tau[l])\big).$$

2. If $\boldsymbol{F}v \in \mathbf{A}$, then $\boldsymbol{T}v \notin D_{\{TLU\uparrow\}}(\Pi, \mathbf{A})$. That is, $|\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\}| < j$ or $|\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{f}l \in \mathbf{A}\}| < n - k$. In view of $|\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\}| + |\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{f}l \in \mathbf{A}\}| = n$, $|\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{f}l \in \mathbf{A}\}| < n - k$ yields $k < |\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\}|$. For $L' = \{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\}$, we have thus shown that $|L'| < j$ or $k < |L'|$. Since $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models \big((\bigwedge_{l \in L'} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L'} \tau[l])\big)$, we conclude that

$$\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models \bigwedge_{\substack{L \subseteq \{l_1,\ldots,l_n\}, \\ |L|<j \text{ or } k<|L|}} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L} \tau[l])\big).$$

We have thus shown that $\boldsymbol{T}v \in \mathbf{A}$ and $\boldsymbol{F}v \in \mathbf{A}$ imply $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[v]$ and $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models \tau[v]$, respectively. That is, $\boldsymbol{T}v \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[v]$. □

Lemma B.8. *Let Π be a disjunctive program and $\mathcal{T}$ a tableau calculus such that $\{TC\uparrow, FC\uparrow\} \subseteq \mathcal{T}$.*

If $card(\Pi) = \emptyset$ or $\{TLU\uparrow, FL\uparrow, FU\uparrow\} \subseteq \mathcal{T}$, then for every non-contradictory complete branch $(\Pi, \mathbf{A})$ and every $v \in conj(\Pi)$, we have that $\boldsymbol{T}v \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[v]$.

Proof. Consider any $v = \{l_1, \ldots, l_n\} \in conj(\Pi)$ and any non-contradictory complete branch $(\Pi, \mathbf{A})$, and assume that $card(\Pi) = \emptyset$ or $\{TLU\uparrow, FL\uparrow, FU\uparrow\} \subseteq \mathcal{T}$. For every $l \in \{l_1, \ldots, l_n\}$, we have that $l \in atom(\Pi) \cup card(\Pi)$ or $l = not\ \pi$ and $\tau[l] = \neg\tau[\pi]$ for some $\pi \in atom(\Pi) \cup card(\Pi)$. By Lemma B.6 and B.7, $\boldsymbol{t}l \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[l]$, and $\boldsymbol{f}l \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models \tau[l]$. We further consider the cases that $\boldsymbol{T}v \in \mathbf{A}$ and $\boldsymbol{F}v \in \mathbf{A}$, respectively:

1. If $\boldsymbol{T}v \in \mathbf{A}$, then $\boldsymbol{F}v \notin D_{\{FC\uparrow\}}(\Pi, \mathbf{A})$. That is, $\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{f}l \in \mathbf{A}\} = \emptyset$ and $\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\} = \{l_1, \ldots, l_n\}$, so that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models (\tau[l_1] \wedge \cdots \wedge \tau[l_n])$.

2. If $\boldsymbol{F}v \in \mathbf{A}$, then $\boldsymbol{T}v \notin D_{\{TC\uparrow\}}(\Pi, \mathbf{A})$. That is, $\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\} \neq \{l_1, \ldots, l_n\}$ and $\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{f}l \in \mathbf{A}\} \neq \emptyset$, so that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models (\tau[l_1] \wedge \cdots \wedge \tau[l_n])$.

We have thus shown that $\boldsymbol{T}v \in \mathbf{A}$ and $\boldsymbol{F}v \in \mathbf{A}$ imply $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[v]$ and $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models \tau[v]$, respectively. That is, $\boldsymbol{T}v \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[v]$. □

Lemma B.9. *Let Π be a disjunctive program and $\mathcal{T}$ a tableau calculus such that $\{TD\uparrow, FD\uparrow\} \subseteq \mathcal{T}$.*

Then, for every non-contradictory complete branch $(\Pi, \mathbf{A})$ and every $v \in disj(\Pi)$, we have that $\boldsymbol{T}v \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[v]$.

Proof. Consider any $v = \{l_1; \ldots; l_n\} \in disj(\Pi)$ and any non-contradictory complete branch $(\Pi, \mathbf{A})$. For every $l \in \{l_1, \ldots, l_n\}$, we have that $l \in atom(\Pi)$ or $l = not\ p$ for some $p \in atom(\Pi)$. By Lemma B.6, $\boldsymbol{t}l \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[l]$, and $\boldsymbol{f}l \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models \tau[l]$. We further consider the cases that $\boldsymbol{T}v \in \mathbf{A}$ and $\boldsymbol{F}v \in \mathbf{A}$, respectively:

1. If $\boldsymbol{T}v \in \mathbf{A}$, then $\boldsymbol{F}v \notin D_{\{FD\uparrow\}}(\Pi, \mathbf{A})$. That is, $\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{f}l \in \mathbf{A}\} \neq \{l_1, \ldots, l_n\}$ and $\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\} \neq \emptyset$, so that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models (\tau[l_1] \vee \cdots \vee \tau[l_n])$.
2. If $\boldsymbol{F}v \in \mathbf{A}$, then $\boldsymbol{T}v \notin D_{\{TD\uparrow\}}(\Pi, \mathbf{A})$. That is, $\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\} = \emptyset$ and $\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{f}l \in \mathbf{A}\} = \{l_1, \ldots, l_n\}$, so that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models (\tau[l_1] \vee \cdots \vee \tau[l_n])$.

We have thus shown that $\boldsymbol{T}v \in \mathbf{A}$ and $\boldsymbol{F}v \in \mathbf{A}$ imply $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[v]$ and $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \not\models \tau[v]$, respectively. That is, $\boldsymbol{T}v \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[v]$. □

Lemmas on Completeness

In order to abstract from the language constructs admitted in a program, the following definition formulates conditions under which we call $\overleftarrow{sup}$, $\overrightarrow{sup}$, min, and max, respectively, *well-behaved*. We then proceed by showing that these four concepts are well-behaved for disjunctive programs.

Definition B.1. *Let α be a literal.*

Then, we define $\overleftarrow{sup}$, $\overrightarrow{sup}$, min, and max, respectively, as well-behaved for α if, for every $S \subseteq \mathcal{P}$ and every assignment $\mathbf{A}$, we have that

1. *if $\overleftarrow{sup}_{\mathbf{A}}(\alpha, S)$ holds, then $\overleftarrow{sup}_{\mathbf{A}'}(\alpha, S)$ holds for every $\mathbf{A}' \subseteq \mathbf{A}$;*
2. *if $\overrightarrow{sup}_{\mathbf{A}}(\alpha, S)$ holds, then $\overrightarrow{sup}_{\mathbf{A}'}(\alpha, S')$ holds for every $\mathbf{A}' \subseteq \mathbf{A}$ and every $S' \subseteq S$;*
3. *if $\sigma \in min_{\mathbf{A}}(\alpha, S)$, then $\overleftarrow{sup}_{\mathbf{A} \cup \{\overline{\sigma}\}}(\alpha, S)$ does not hold;*
4. *if $\sigma \in max_{\mathbf{A}}(\alpha, S)$, then $\overrightarrow{sup}_{\mathbf{A} \cup \{\overline{\sigma}\}}(\alpha, S)$ does not hold.*

Lemma B.10. *Let α be a disjunctive literal and β a cardinality literal or a possibly negated conjunction of cardinality literals.*

Then, we have that $\overleftarrow{sup}$ and min are well-behaved for α and that $\overrightarrow{sup}$ and max are well-behaved for β.

Proof. Let $S \subseteq \mathcal{P}$ and $\mathbf{A}$ an arbitrary assignment.

We first consider the possible cases such that $\overleftarrow{sup}_{\mathbf{A}}(\alpha, S)$ holds:

1. If $\alpha \in S$, we have that $\overleftarrow{sup}_{\mathbf{A}'}(\alpha, S)$ holds for every assignment $\mathbf{A}'$.

2. If $\alpha = j\{l_1, \ldots, l_n\}k \in \mathit{card}(\mathcal{P})$, then $\{l_1, \ldots, l_n\} \cap S \neq \emptyset$ and $|\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\}| < k$. Since for every $\mathbf{A}' \subseteq \mathbf{A}$, we have that $|\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}'\}| \leq |\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\}| < k$, we conclude that $\overleftarrow{\mathit{sup}}_{\mathbf{A}'}(\alpha, S)$ holds.

3. If $\alpha = \{l_1; \ldots; l_n\} \in \mathit{disj}(\mathcal{P})$, then $\{l_1, \ldots, l_n\} \cap S \neq \emptyset$ and $\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\} = \emptyset$. Since for every $\mathbf{A}' \subseteq \mathbf{A}$, we have that $\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}'\} \subseteq \{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\} = \emptyset$, we conclude that $\overleftarrow{\mathit{sup}}_{\mathbf{A}'}(\alpha, S)$ holds.

We next consider the possible cases such that $\sigma \in \mathit{min}_{\mathbf{A}}(\alpha, S)$:

1. If $\alpha = j\{l_1, \ldots, l_n\}k \in \mathit{card}(\mathcal{P})$ and $\sigma \in \mathit{min}_{\mathbf{A}}(\alpha, S) = \{\boldsymbol{f}l \mid l \in \{l_1, \ldots, l_n\} \setminus S, \boldsymbol{t}l \notin \mathbf{A}\}$, then $|\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\}| = k - 1$. That is, $\overline{\sigma} = \boldsymbol{t}l \notin \mathbf{A}$ for some $l \in \{l_1, \ldots, l_n\} \setminus S$, so that $|\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A} \cup \{\overline{\sigma}\}\}| = |\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\}| + 1 = k$, which means that $\overleftarrow{\mathit{sup}}_{\mathbf{A} \cup \{\overline{\sigma}\}}(\alpha, S)$ does not hold.

2. If $\alpha = \{l_1; \ldots; l_n\} \in \mathit{disj}(\mathcal{P})$ and $\sigma \in \mathit{min}_{\mathbf{A}}(\alpha, S) = \{\boldsymbol{f}l \mid l \in \{l_1, \ldots, l_n\} \setminus S\}$, then $\overline{\sigma} = \boldsymbol{t}l$ for some $l \in \{l_1, \ldots, l_n\} \setminus S$. We conclude that $\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A} \cup \{\overline{\sigma}\}\} \neq \emptyset$, which means that $\overleftarrow{\mathit{sup}}_{\mathbf{A} \cup \{\overline{\sigma}\}}(\alpha, S)$ does not hold.

We now come to the possible cases such that $\overrightarrow{\mathit{sup}}_{\mathbf{A}}(\beta, S)$ holds:

1. If $\beta = \mathit{not}\ v$, where $v \in \mathcal{P} \cup \mathit{card}(\mathcal{P}) \cup \mathit{conj}(\mathcal{P})$, we have that $\overrightarrow{\mathit{sup}}_{\mathbf{A}'}(\beta, S')$ holds for every assignment $\mathbf{A}'$ and every $S' \subseteq \mathcal{P}$.

2. If $\beta \in \mathcal{P} \setminus S$, then $\beta \in \mathcal{P} \setminus S'$ for every $S' \subseteq S$, so that $\overrightarrow{\mathit{sup}}_{\mathbf{A}'}(\beta, S')$ holds for every assignment $\mathbf{A}'$ and every $S' \subseteq S$.

3. If $\beta = j\{l_1, \ldots, l_n\}k \in \mathit{card}(\mathcal{P})$, then $|\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{f}l \notin \mathbf{A}\}| \geq j$. Since for every $\mathbf{A}' \subseteq \mathbf{A}$ and every $S' \subseteq S$, we have that $|\{l \in \{l_1, \ldots, l_n\} \setminus S' \mid \boldsymbol{f}l \notin \mathbf{A}'\}| \geq |\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{f}l \notin \mathbf{A}\}| \geq j$, we conclude that $\overrightarrow{\mathit{sup}}_{\mathbf{A}'}(\beta, S')$ holds.

4. If $\beta = \{l_1, \ldots, l_n\} \in \mathit{conj}(\mathcal{P})$, then $\overrightarrow{\mathit{sup}}_{\mathbf{A}}(l, S)$ holds for every $l \in \{l_1, \ldots, l_n\}$. Furthermore, since one of the first three cases applies to each $l \in \{l_1, \ldots, l_n\}$, we have that $\overrightarrow{\mathit{sup}}_{\mathbf{A}'}(l, S')$ holds for every $\mathbf{A}' \subseteq \mathbf{A}$ and every $S' \subseteq S$, so that $\overrightarrow{\mathit{sup}}_{\mathbf{A}'}(\beta, S')$ holds as well.

Finally, we consider the possible cases such that $\sigma \in \mathit{max}_{\mathbf{A}}(\beta, S)$:

1. If $\beta = j\{l_1, \ldots, l_n\}k \in \mathit{card}(\mathcal{P})$ and $\sigma \in \mathit{max}_{\mathbf{A}}(\beta, S) = \{\boldsymbol{t}l \mid l \in \{l_1, \ldots, l_n\} \setminus S, \boldsymbol{f}l \notin \mathbf{A}\}$, then $|\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{f}l \notin \mathbf{A}\}| = j$. That is, $\overline{\sigma} = \boldsymbol{f}l \notin \mathbf{A}$ for some $l \in \{l_1, \ldots, l_n\} \setminus S$, so that $|\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{f}l \notin \mathbf{A} \cup \{\overline{\sigma}\}\}| = |\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{f}l \notin \mathbf{A}\}| - 1 = j - 1$, which means that $\overrightarrow{\mathit{sup}}_{\mathbf{A} \cup \{\overline{\sigma}\}}(\beta, S)$ does not hold.

2. If $\beta = \{l_1, \ldots, l_n\} \in \mathit{conj}(\mathcal{P})$ and $\sigma \in \mathit{max}_{\mathbf{A}}(\beta, S) = \bigcup_{l \in \{l_1, \ldots, l_n\}} \mathit{max}_{\mathbf{A}}(l, S)$, then $\sigma \in \mathit{max}_{\mathbf{A}}(l, S)$ for some $l \in \{l_1, \ldots, l_n\} \cap \mathit{card}(\mathcal{P})$. That is, the previous case applies to l, so that $\overrightarrow{\mathit{sup}}_{\mathbf{A} \cup \{\overline{\sigma}\}}(l, S)$ and $\overrightarrow{\mathit{sup}}_{\mathbf{A} \cup \{\overline{\sigma}\}}(\beta, S)$ do not hold.

We have thus, for $S \subseteq \mathcal{P}$ and an arbitrary assignment $\mathbf{A}$, considered all possible cases and shown that $\overleftarrow{\mathit{sup}}$ and min are well-behaved for α and that $\overrightarrow{\mathit{sup}}$ and max are well-behaved for β. □

The concept of well-behavedness allows us to identify the property that $sup_{\mathbf{A}}(\Pi, S, T)$ is anti-monotone w.r.t. both $\mathbf{A}$ and T.

Lemma B.11. *Let Π be a disjunctive program, $S \subseteq \mathcal{P}$, $T \subseteq \mathcal{P}$, and $\mathbf{A}$ an assignment.*

If $\overleftarrow{sup}$ and $\overrightarrow{sup}$ are well-behaved for all literals in $\{\alpha \mid (\alpha \leftarrow \beta) \in \Pi\}$ and $\{\beta \mid (\alpha \leftarrow \beta) \in \Pi\}$, respectively, then we have that $sup_{\mathbf{A}}(\Pi, S, T) \subseteq sup_{\mathbf{A}'}(\Pi, S, T')$ for every $\mathbf{A}' \subseteq \mathbf{A}$ and every $T' \subseteq T$.

Proof. Assume that $\overleftarrow{sup}$ and $\overrightarrow{sup}$ are well-behaved for all literals in $\{\alpha \mid (\alpha \leftarrow \beta) \in \Pi\}$ and $\{\beta \mid (\alpha \leftarrow \beta) \in \Pi\}$, respectively, and consider any $(\alpha \leftarrow \beta) \in sup_{\mathbf{A}}(\Pi, S, T) = \{(\alpha \leftarrow \beta) \in \Pi \mid \boldsymbol{f}\beta \notin \mathbf{A}, \overleftarrow{sup}_{\mathbf{A}}(\alpha, S), \overrightarrow{sup}_{\mathbf{A}}(\beta, T)\}$. In view of Definition B.1, for every $\mathbf{A}' \subseteq \mathbf{A}$ and every $T' \subseteq T$, we have that $\overleftarrow{sup}_{\mathbf{A}}(\alpha, S)$ and $\overrightarrow{sup}_{\mathbf{A}}(\beta, T)$ imply $\overleftarrow{sup}_{\mathbf{A}'}(\alpha, S)$ and $\overrightarrow{sup}_{\mathbf{A}'}(\beta, T')$, respectively, and $\boldsymbol{f}\beta \notin \mathbf{A}'$ follows immediately from $\boldsymbol{f}\beta \notin \mathbf{A}$. From this, we conclude that $(\alpha \leftarrow \beta) \in sup_{\mathbf{A}'}(\Pi, S, T') = \{(\alpha \leftarrow \beta) \in \Pi \mid \boldsymbol{f}\beta \notin \mathbf{A}', \overleftarrow{sup}_{\mathbf{A}'}(\alpha, S), \overrightarrow{sup}_{\mathbf{A}'}(\beta, T')\}$. □

We are now ready to prove that, for a total assignment $\mathbf{A}$ such that the deterministic tableau rules in Figure 3.4 on Page 34 do not yield a contradiction, the entries of $\mathbf{A}$ are preserved when applying these tableau rules w.r.t. any assignment contained in $\mathbf{A}$.

Lemma B.12. *Let Π be a disjunctive program and $\mathbf{A}$ a total assignment such that $\boldsymbol{t}\beta \notin \mathbf{A}$ or $\boldsymbol{f}\alpha \notin \mathbf{A}$ for every $(\alpha \leftarrow \beta) \in \Pi$ and $sup_{\mathbf{A}}(\Pi, S, S) \neq \emptyset$ or $\mathbf{A}^{\boldsymbol{T}} \cap S = \emptyset$ for every $S \subseteq atom(\Pi)$.*

If $\overleftarrow{sup}$ and min are well-behaved for all literals in $\{\alpha \mid (\alpha \leftarrow \beta) \in \Pi\}$ and if $\overrightarrow{sup}$ and max are well-behaved for all literals in $\{\beta \mid (\alpha \leftarrow \beta) \in \Pi\}$, then for every $\mathbf{A}' \subseteq \mathbf{A}$, we have that $D_{\{(a)\text{–}(f)\}}(\Pi, \mathbf{A}') \subseteq \mathbf{A}$.

Proof. Assume that $\overleftarrow{sup}$ and min are well-behaved for all literals in $\{\alpha \mid (\alpha \leftarrow \beta) \in \Pi\}$ and that $\overrightarrow{sup}$ and max are well-behaved for all literals in $\{\beta \mid (\alpha \leftarrow \beta) \in \Pi\}$, and consider any $\mathbf{A}' \subseteq \mathbf{A}$. We show that any entry deducible by $I\uparrow$, $I\downarrow$, $N\uparrow$, $N\downarrow$, $U\uparrow$, or $U\downarrow$ in $(\Pi, \mathbf{A}')$ belongs to $\mathbf{A}$:

($I\uparrow$) If $\boldsymbol{t}\alpha \in D_{\{I\uparrow\}}(\Pi, \mathbf{A}')$, we have that $\boldsymbol{t}\beta \in \mathbf{A}'$ for some $(\alpha \leftarrow \beta) \in \Pi$. Since $\boldsymbol{t}\beta \in \mathbf{A}$, it holds that $\boldsymbol{f}\alpha \notin \mathbf{A}$, which yields $\boldsymbol{t}\alpha \in \mathbf{A}$ because $\mathbf{A}$ is total.

($I\downarrow$) If $\boldsymbol{f}\beta \in D_{\{I\downarrow\}}(\Pi, \mathbf{A}')$, we have that $\boldsymbol{f}\alpha \in \mathbf{A}'$ for some $(\alpha \leftarrow \beta) \in \Pi$. Since $\boldsymbol{f}\alpha \in \mathbf{A}$, it holds that $\boldsymbol{t}\beta \notin \mathbf{A}$, which yields $\boldsymbol{f}\beta \in \mathbf{A}$ because $\mathbf{A}$ is total.

($N\uparrow$) If $\boldsymbol{F}p \in D_{\{N\uparrow\}}(\Pi, \mathbf{A}')$, we have that $p \in atom(\Pi)$ and $sup_{\mathbf{A}'}(\Pi, \{p\}, \emptyset) = \emptyset$. By Lemma B.11, we conclude that $sup_{\mathbf{A}}(\Pi, \{p\}, \{p\}) = \emptyset$. Thus, it holds that $\boldsymbol{T}p \notin \mathbf{A}$, which yields $\boldsymbol{F}p \in \mathbf{A}$ because $\mathbf{A}$ is total.

($N\downarrow$) If $\sigma \in D_{\{N\downarrow\}}(\Pi, \mathbf{A}')$, we have that $\sigma \in \{\boldsymbol{t}\beta\} \cup min_{\mathbf{A}'}(\alpha, \{p\}) \cup max_{\mathbf{A}'}(\beta, \emptyset)$ for some $p \in (\mathbf{A}')^{\boldsymbol{T}} \cap atom(\Pi)$ such that $sup_{\mathbf{A}'}(\Pi, \{p\}, \emptyset) = \{\alpha \leftarrow \beta\}$. Since $p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$, it holds that $sup_{\mathbf{A}}(\Pi, \{p\}, \{p\}) \neq \emptyset$. However, given that min and max are well-behaved for α and β, respectively, we also have that $(\alpha \leftarrow \beta) \notin sup_{\mathbf{A}' \cup \{\overline{\sigma}\}}(\Pi, \{p\}, \emptyset)$. By Lemma B.11, we conclude that $sup_{\mathbf{A} \cup \{\overline{\sigma}\}}(\Pi, \{p\}, \{p\}) \subseteq sup_{\mathbf{A}' \cup \{\overline{\sigma}\}}(\Pi, \{p\}, \emptyset) \subseteq sup_{\mathbf{A}'}(\Pi, \{p\}, \emptyset) \setminus \{\alpha \leftarrow \beta\} = \emptyset$. That is, $sup_{\mathbf{A} \cup \{\overline{\sigma}\}}(\Pi, \{p\}, \{p\}) = \emptyset \neq sup_{\mathbf{A}}(\Pi, \{p\}, \{p\})$, which yields $\overline{\sigma} \notin \mathbf{A}$. Finally, since $\mathbf{A}$ is total, $\overline{\sigma} \notin \mathbf{A}$ implies $\sigma \in \mathbf{A}$.

($U\uparrow$) If $\boldsymbol{F}p \in D_{\{U\uparrow\}}(\Pi, \mathbf{A}')$, we have that $p \in S$ for some $S \subseteq atom(\Pi)$ such that $sup_{\mathbf{A}'}(\Pi, S, S) = \emptyset$. By Lemma B.11, we conclude that $sup_{\mathbf{A}}(\Pi, S, S) = \emptyset$. Thus, it holds that $\boldsymbol{T}p \notin \mathbf{A}$, which yields $\boldsymbol{F}p \in \mathbf{A}$ because $\mathbf{A}$ is total.

($U\downarrow$) If $\sigma \in D_{\{U\downarrow\}}(\Pi, \mathbf{A}')$, we have that $\sigma \in \{\boldsymbol{t}\beta\} \cup min_{\mathbf{A}'}(\alpha, S) \cup max_{\mathbf{A}'}(\beta, S)$ for some $S \subseteq atom(\Pi)$ such that $(\mathbf{A}')^{\boldsymbol{T}} \cap S \neq \emptyset$ and $sup_{\mathbf{A}'}(\Pi, S, S) = \{\alpha \leftarrow \beta\}$. Since $\mathbf{A}^{\boldsymbol{T}} \cap S \neq \emptyset$, it holds that $sup_{\mathbf{A}}(\Pi, S, S) \neq \emptyset$. However, given that min and max are well-behaved for α and β, respectively, we also have that $(\alpha \leftarrow \beta) \notin sup_{\mathbf{A}'\cup\{\overline{\sigma}\}}(\Pi, S, S)$. By Lemma B.11, we conclude that $sup_{\mathbf{A}\cup\{\overline{\sigma}\}}(\Pi, S, S) \subseteq sup_{\mathbf{A}'\cup\{\overline{\sigma}\}}(\Pi, S, S) \subseteq sup_{\mathbf{A}'}(\Pi, S, S) \setminus \{\alpha \leftarrow \beta\} = \emptyset$. That is, $sup_{\mathbf{A}\cup\{\overline{\sigma}\}}(\Pi, S, S) = \emptyset \neq sup_{\mathbf{A}}(\Pi, S, S)$, which yields $\overline{\sigma} \notin \mathbf{A}$. Finally, since $\mathbf{A}$ is total, $\overline{\sigma} \notin \mathbf{A}$ implies $\sigma \in \mathbf{A}$.

We have thus shown that, in every branch $(\Pi, \mathbf{A}')$ such that $\mathbf{A}' \subseteq \mathbf{A}$, any entry deducible by $I\uparrow$, $I\downarrow$, $N\uparrow$, $N\downarrow$, $U\uparrow$, or $U\downarrow$ belongs to $\mathbf{A}$, so that $D_{\{(a)\text{–}(f)\}}(\Pi, \mathbf{A}') \subseteq \mathbf{A}$. □

Finally, the next two lemmas show that, for a total assignment $\mathbf{A}$ such that the truth values of variables $v \in atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \cup disj(\Pi)$ match the valuation of $\tau[v]$ w.r.t. $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$, the language-specific tableau rules in Figure 3.5, 3.7, and 3.8 on Page 37, 41, and 44, respectively, preserve the entries of $\mathbf{A}$ when applied w.r.t. any assignment contained in $\mathbf{A}$.

Lemma B.13. *Let Π be a disjunctive program, $X \subseteq atom(\Pi)$, and*

$$\begin{aligned}\mathbf{A} &= \{\boldsymbol{T}v \mid v \in atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \cup disj(\Pi), X \models \tau[v]\} \\ &\cup \{\boldsymbol{F}v \mid v \in atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \cup disj(\Pi), X \not\models \tau[v]\}.\end{aligned}$$

Then, for every $v \in atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \cup disj(\Pi)$, we have that

1. $\boldsymbol{t}v \in \mathbf{A}$ *iff* $X \models \tau[v]$;
2. $\boldsymbol{t}not\ v \in \mathbf{A}$ *iff* $X \models \tau[not\ v]$;
3. $\boldsymbol{f}v \in \mathbf{A}$ *iff* $X \not\models \tau[v]$;
4. $\boldsymbol{f}not\ v \in \mathbf{A}$ *iff* $X \not\models \tau[not\ v]$.

Proof. By the definition of $\mathbf{A}$, for every $v \in atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \cup disj(\Pi)$:

1. $\boldsymbol{t}v \in \mathbf{A}$ iff $\boldsymbol{T}v \in \mathbf{A}$ iff $X \models \tau[v]$;
2. $\boldsymbol{t}not\ v \in \mathbf{A}$ iff $\boldsymbol{F}v \in \mathbf{A}$ iff $X \not\models \tau[v]$ iff $X \models \neg\tau[v]$ iff $X \models \tau[not\ v]$;
3. $\boldsymbol{f}v \in \mathbf{A}$ iff $\boldsymbol{F}v \in \mathbf{A}$ iff $X \not\models \tau[v]$;
4. $\boldsymbol{f}not\ v \in \mathbf{A}$ iff $\boldsymbol{T}v \in \mathbf{A}$ iff $X \models \tau[v]$ iff $X \not\models \neg\tau[v]$ iff $X \not\models \tau[not\ v]$.

We have thus shown that all items of the statement hold. □

Lemma B.14. *Let Π be a disjunctive program, $X \subseteq atom(\Pi)$, and*

$$\begin{aligned}\mathbf{A} &= \{\boldsymbol{T}v \mid v \in atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \cup disj(\Pi), X \models \tau[v]\} \\ &\cup \{\boldsymbol{F}v \mid v \in atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \cup disj(\Pi), X \not\models \tau[v]\}.\end{aligned}$$

Then, for every $\mathbf{A}' \subseteq \mathbf{A}$, we have that $D_{\{(h)\text{–}(v)\}}(\Pi, \mathbf{A}') \subseteq \mathbf{A}$.

Proof. By Lemma B.13, for every literal $l = v$ or $l = not\ v$, where $v \in atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \cup disj(\Pi)$, we have that $\boldsymbol{t}l \in \mathbf{A}$ iff $X \models \tau[l]$, and that $\boldsymbol{f}l \in \mathbf{A}$ iff $X \not\models \tau[l]$. Hence, we can treat such conditions as synonyms in the following consideration of some $\mathbf{A}' \subseteq \mathbf{A}$ and the tableau rules *(h)–(v)*:

($TC\uparrow$) If $\{l_1, \ldots, l_n\} \in conj(\Pi)$ such that $\{\boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\} \subseteq \mathbf{A}'$, we have that $X \models \tau[l_1], \ldots, X \models \tau[l_n]$. That is, $X \models (\tau[l_1] \wedge \cdots \wedge \tau[l_n])$, so that $\boldsymbol{T}\{l_1, \ldots, l_n\} \in \mathbf{A}$.

($TC\downarrow$) If $\{l_1, \ldots, l_{i-1}, l_i, l_{i+1}, \ldots, l_n\} \in conj(\Pi)$ such that $\{\boldsymbol{F}\{l_1, \ldots, l_{i-1}, l_i, l_{i+1}, \ldots, l_n\}, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_{i-1}, \boldsymbol{t}l_{i+1}, \ldots, \boldsymbol{t}l_n\} \subseteq \mathbf{A}'$, we have that $X \models \tau[l_1], \ldots, X \models \tau[l_{i-1}], X \models \tau[l_{i+1}], \ldots, X \models \tau[l_n]$ but $X \not\models (\tau[l_1] \wedge \cdots \wedge \tau[l_{i-1}] \wedge \tau[l_i] \wedge \tau[l_{i+1}] \wedge \cdots \wedge \tau[l_n])$. That is, $X \not\models \tau[l_i]$, so that $\boldsymbol{f}l_i \in \mathbf{A}$.

($FC\uparrow$) If $\{l_1, \ldots, l_i, \ldots, l_n\} \in conj(\Pi)$ such that $\boldsymbol{f}l_i \in \mathbf{A}'$, we have that $X \not\models \tau[l_i]$. That is, $X \not\models (\tau[l_1] \wedge \cdots \wedge \tau[l_i] \wedge \cdots \wedge \tau[l_n])$, so that $\boldsymbol{F}\{l_1, \ldots, l_i, \ldots, l_n\} \in \mathbf{A}$.

($FC\downarrow$) If $\{l_1, \ldots, l_n\} \in conj(\Pi)$ such that $\boldsymbol{T}\{l_1, \ldots, l_n\} \in \mathbf{A}'$, we have that $X \models (\tau[l_1] \wedge \cdots \wedge \tau[l_n])$. That is, $X \models \tau[l_1], \ldots, X \models \tau[l_n]$, so that $\{\boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\} \subseteq \mathbf{A}$.

($TLU\uparrow$) If $j\{l_1, \ldots, l_j, \ldots, l_{k+1}, \ldots, l_n\}k \in card(\Pi)$ such that $\{\boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_j, \boldsymbol{f}l_{k+1}, \ldots, \boldsymbol{f}l_n\} \subseteq \mathbf{A}'$, for any $L \subseteq \{l_1, \ldots, l_n\}$ such that $|L| < j$, we have that $\{l_1, \ldots, l_j\} \not\subseteq L$, that is, $X \models (\bigvee_{l \in \{l_1, \ldots, l_n\} \setminus L} \tau[l])$. Furthermore, for any $L \subseteq \{l_1, \ldots, l_n\}$ such that $k < |L|$, we have that $L \cap \{l_{k+1}, \ldots, l_n\} \neq \emptyset$, that is, $X \not\models (\bigwedge_{l \in L} \tau[l])$. We obtain that

$$X \models \bigwedge_{L \subseteq \{l_1, \ldots, l_n\}, |L| < j \text{ or } k < |L|} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1, \ldots, l_n\} \setminus L} \tau[l])\big),$$

so that $\boldsymbol{T}j\{l_1, \ldots, l_j, \ldots, l_{k+1}, \ldots, l_n\}k \in \mathbf{A}$.

($TLU\downarrow$) If $j\{l_1, \ldots, l_{j-1}, l_j, \ldots, l_k, l_{k+1}, \ldots, l_n\}k \in card(\Pi)$ such that $\{\boldsymbol{F}j\{l_1, \ldots, l_{j-1}, l_j, \ldots, l_k, l_{k+1}, \ldots, l_n\}k, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_{j-1}, \boldsymbol{f}l_{k+1}, \ldots, \boldsymbol{f}l_n\} \subseteq \mathbf{A}'$, we have that

$$X \not\models \bigwedge_{L \subseteq \{l_1, \ldots, l_n\}, |L| < j \text{ or } k < |L|} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1, \ldots, l_n\} \setminus L} \tau[l])\big).$$

However, for any $L \subseteq \{l_1, \ldots, l_n\}$ such that $k < |L|$, we have that $L \cap \{l_{k+1}, \ldots, l_n\} \neq \emptyset$, that is, $X \not\models (\bigwedge_{l \in L} \tau[l])$. Furthermore, for any $L \subseteq \{l_1, \ldots, l_n\}$ such that $|L| < j$ and $L \neq \{l_1, \ldots, l_{j-1}\}$, we have that $\{l_1, \ldots, l_{j-1}\} \not\subseteq L$, that is, $X \models (\bigvee_{l \in \{l_1, \ldots, l_n\} \setminus L} \tau[l])$. We obtain that

$$X \models \bigwedge_{\substack{L \subseteq \{l_1, \ldots, l_n\}, \\ (|L| < j \text{ and } L \neq \{l_1, \ldots, l_{j-1}\}) \text{ or } k < |L|}} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1, \ldots, l_n\} \setminus L} \tau[l])\big).$$

That is, $X \not\models \big((\tau[l_1] \wedge \cdots \wedge \tau[l_{j-1}]) \rightarrow (\tau[l_j] \vee \cdots \vee \tau[l_k] \vee \tau[l_{k+1}] \vee \cdots \vee \tau[l_n])\big)$, so that $X \not\models \tau[l_j], \ldots, X \not\models \tau[l_k]$ and $\{\boldsymbol{f}l_j, \ldots, \boldsymbol{f}l_k\} \subseteq \mathbf{A}$.

($TLU\downarrow$) If $j\{l_1, \ldots, l_j, l_{j+1}, \ldots, l_{k+1}, l_{k+2}, \ldots, l_n\}k \in card(\Pi)$ such that $\{\boldsymbol{F}j\{l_1, \ldots, l_j, l_{j+1}, \ldots, l_{k+1}, l_{k+2}, \ldots, l_n\}k, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_j, \boldsymbol{f}l_{k+2}, \ldots, \boldsymbol{f}l_n\} \subseteq \mathbf{A}'$, we have that

$$X \not\models \bigwedge_{L \subseteq \{l_1, \ldots, l_n\}, |L| < j \text{ or } k < |L|} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1, \ldots, l_n\} \setminus L} \tau[l])\big).$$

However, for any $L \subseteq \{l_1, \ldots, l_n\}$ such that $|L| < j$, we have that $\{l_1, \ldots, l_j\} \not\subseteq L$, that is, $X \models (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L} \tau[l])$. Furthermore, for any $L \subseteq \{l_1, \ldots, l_n\}$ such that $k < |L|$ and $L \neq \{l_1, \ldots, l_{k+1}\}$, we have that $L \cap \{l_{k+2}, \ldots, l_n\} \neq \emptyset$, that is, $X \not\models (\bigwedge_{l \in L} \tau[l])$. We obtain that

$$X \models \bigwedge_{\substack{L \subseteq \{l_1,\ldots,l_n\}, \\ |L|<j \text{ or } (k<|L| \text{ and } L \neq \{l_1,\ldots,l_{k+1}\})}} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L} \tau[l])\big).$$

That is, $X \not\models \big((\tau[l_1] \wedge \cdots \wedge \tau[l_j] \wedge \tau[l_{j+1}] \wedge \cdots \wedge \tau[l_{k+1}]) \rightarrow (\tau[l_{k+2}] \vee \cdots \vee \tau[l_n])\big)$, so that $X \models \tau[l_{j+1}], \ldots, X \models \tau[l_{k+1}]$ and $\{\boldsymbol{t} l_{j+1}, \ldots, \boldsymbol{t} l_{k+1}\} \subseteq \mathbf{A}$.

(*FL*↑) If $j\{l_1, \ldots, l_j, \ldots, l_n\}k \in \mathit{card}(\Pi)$ such that $\{\boldsymbol{f} l_j, \ldots, \boldsymbol{f} l_n\} \subseteq \mathbf{A}'$, for $L' = \{l \in \{l_1, \ldots, l_n\} \mid X \models \tau[l]\}$, we have that $L' \subseteq \{l_1, \ldots, l_{j-1}\}$ and $|L'| < j$, while $X \not\models \tau[l]$ for all $l \in \{l_1, \ldots, l_n\} \setminus L'$. Hence, $X \not\models \big((\bigwedge_{l \in L'} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L'} \tau[l])\big)$ and

$$X \not\models \bigwedge_{L \subseteq \{l_1,\ldots,l_n\}, |L|<j \text{ or } k<|L|} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L} \tau[l])\big),$$

so that $\boldsymbol{F} j\{l_1, \ldots, l_j, \ldots, l_n\}k \in \mathbf{A}$.

(*FL*↓) If $j\{l_1, \ldots, l_j, l_{j+1}, \ldots, l_n\}k \in \mathit{card}(\Pi)$ such that $\{\boldsymbol{T} j\{l_1, \ldots, l_j, l_{j+1}, \ldots, l_n\}k, \boldsymbol{f} l_{j+1}, \ldots, \boldsymbol{f} l_n\} \subseteq \mathbf{A}'$, we have that

$$X \models \bigwedge_{L \subseteq \{l_1,\ldots,l_n\}, |L|<j \text{ or } k<|L|} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L} \tau[l])\big).$$

In particular, for every $1 \leq i \leq j$ and $L_i = \{l \in \{l_1, \ldots, l_n\} \setminus \{l_i\} \mid X \models \tau[l]\}$, we have that $L_i \subseteq \{l_1, \ldots, l_j\} \setminus \{l_i\}$ and $|L_i| < j$, while $X \not\models \tau[l]$ for every $l \in \{l_1, \ldots, l_n\} \setminus (L_i \cup \{l_i\})$. Hence, $X \models \big((\bigwedge_{l \in L_i} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L_i} \tau[l])\big)$ but $X \not\models \big((\bigwedge_{l \in L_i} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus (L_i \cup \{l_i\})} \tau[l])\big)$. That is, $X \models \tau[l_i]$ for every $1 \leq i \leq j$, so that $\{\boldsymbol{t} l_1, \ldots, \boldsymbol{t} l_j\} \subseteq \mathbf{A}$.

(*FU*↑) If $j\{l_1, \ldots, l_{k+1}, \ldots, l_n\}k \in \mathit{card}(\Pi)$ such that $\{\boldsymbol{t} l_1, \ldots, \boldsymbol{t} l_{k+1}\} \subseteq \mathbf{A}'$, for $L' = \{l \in \{l_1, \ldots, l_n\} \mid X \models \tau[l]\}$, we have that $\{l_1, \ldots, l_{k+1}\} \subseteq L'$ and $k < |L'|$, while $X \not\models \tau[l]$ for all $l \in \{l_1, \ldots, l_n\} \setminus L'$. Hence, $X \not\models \big((\bigwedge_{l \in L'} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L'} \tau[l])\big)$ and

$$X \not\models \bigwedge_{L \subseteq \{l_1,\ldots,l_n\}, |L|<j \text{ or } k<|L|} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L} \tau[l])\big),$$

so that $\boldsymbol{F} j\{l_1, \ldots, l_{k+1}, \ldots, l_n\}k \in \mathbf{A}$.

(*FU*↓) If $j\{l_1, \ldots, l_k, l_{k+1}, \ldots, l_n\}k \in \mathit{card}(\Pi)$ such that $\{\boldsymbol{T} j\{l_1, \ldots, l_k, l_{k+1}, \ldots, l_n\}k, \boldsymbol{t} l_1, \ldots, \boldsymbol{t} l_k\} \subseteq \mathbf{A}'$, we have that

$$X \models \bigwedge_{L \subseteq \{l_1,\ldots,l_n\}, |L|<j \text{ or } k<|L|} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L} \tau[l])\big).$$

In particular, for every $k < i \leq n$ and $L_i = \{l \in \{l_1, \ldots, l_n\} \mid X \models \tau[l]\} \cup \{l_i\}$, we have that $\{l_1, \ldots, l_k\} \cup \{l_i\} \subseteq L_i$ and $k < |L_i|$, while $X \not\models \tau[l]$ for every $l \in \{l_1, \ldots, l_n\} \setminus L_i$. Hence, $X \models \big((\bigwedge_{l \in L_i} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L_i} \tau[l])\big)$ but $X \not\models \big((\bigwedge_{l \in L_i \setminus \{l_i\}} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L_i} \tau[l])\big)$. That is, $X \not\models \tau[l_i]$ for every $k < i \leq n$, so that $\{\boldsymbol{f} l_{k+1}, \ldots, \boldsymbol{f} l_n\} \subseteq \mathbf{A}$.

(*TD*↑) If $\{l_1;\ldots;l_i;\ldots;l_n\} \in \mathit{disj}(\Pi)$ such that $\boldsymbol{t}l_i \in \mathbf{A}'$, we have that $X \models \tau[l_i]$. That is, $X \models (\tau[l_1] \vee \cdots \vee \tau[l_i] \vee \cdots \vee \tau[l_n])$, so that $\boldsymbol{T}\{l_1;\ldots;l_i;\ldots;l_n\} \in \mathbf{A}$.

(*TD*↓) If $\{l_1;\ldots;l_n\} \in \mathit{disj}(\Pi)$ such that $\boldsymbol{F}\{l_1;\ldots;l_n\} \in \mathbf{A}'$, we have that $X \not\models (\tau[l_1] \vee \cdots \vee \tau[l_n])$. That is, $X \not\models \tau[l_1], \ldots, X \not\models \tau[l_n]$, so that $\{\boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_n\} \subseteq \mathbf{A}$.

(*FD*↑) If $\{l_1;\ldots;l_n\} \in \mathit{disj}(\Pi)$ such that $\{\boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_n\} \subseteq \mathbf{A}'$, we have that $X \not\models \tau[l_1], \ldots, X \not\models \tau[l_n]$. That is, $X \not\models (\tau[l_1] \vee \cdots \vee \tau[l_n])$, so that $\boldsymbol{F}\{l_1;\ldots;l_n\} \in \mathbf{A}$.

(*FD*↓) If $\{l_1;\ldots;l_{i-1};l_i;l_{i+1};\ldots;l_n\} \in \mathit{disj}(\Pi)$ such that $\{\boldsymbol{T}\{l_1;\ldots;l_{i-1};l_i;l_{i+1};\ldots;l_n\}, \boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_{i-1}, \boldsymbol{f}l_{i+1}, \ldots, \boldsymbol{f}l_n\} \subseteq \mathbf{A}'$, we have that $X \not\models \tau[l_1], \ldots, X \not\models \tau[l_{i-1}], X \not\models \tau[l_{i+1}], \ldots, X \not\models \tau[l_n]$ but $X \models (\tau[l_1] \vee \cdots \vee \tau[l_{i-1}] \vee \tau[l_i] \vee \tau[l_{i+1}] \vee \cdots \vee \tau[l_n])$. That is, $X \models \tau[l_i]$, so that $\boldsymbol{t}l_i \in \mathbf{A}$.

We have thus shown that, in every branch $(\Pi, \mathbf{A}')$ such that $\mathbf{A}' \subseteq \mathbf{A}$, any entry deducible by some of the tableau rules *(h)–(v)* belongs to $\mathbf{A}$, so that $D_{\{(h)\text{–}(v)\}}(\Pi, \mathbf{A}') \subseteq \mathbf{A}$. □

Proofs of Soundness and Completeness

The following theorem characterizes the answer sets of a disjunctive program in terms of total assignments $\mathbf{A}$ such that the generic tableau rules in Figure 3.4 on Page 34 do not yield a contradiction and the entries in $\mathbf{A}$ match the valuations of propositional formulas associated with their variables.

Theorem B.15. *Let* Π *be a disjunctive program and* $X \subseteq \mathit{atom}(\Pi)$.
Then, we have that X *is an answer set of* Π *iff*

$$\begin{aligned}\mathbf{A} \;=\; & \{\boldsymbol{T}v \mid v \in \mathit{atom}(\Pi) \cup \mathit{conj}(\Pi) \cup \mathit{card}(\Pi) \cup \mathit{disj}(\Pi), X \models \tau[v]\} \\ \cup\; & \{\boldsymbol{F}v \mid v \in \mathit{atom}(\Pi) \cup \mathit{conj}(\Pi) \cup \mathit{card}(\Pi) \cup \mathit{disj}(\Pi), X \not\models \tau[v]\}\end{aligned}$$

is such that $\boldsymbol{t}\beta \notin \mathbf{A}$ *or* $\boldsymbol{f}\alpha \notin \mathbf{A}$ *for every* $(\alpha \leftarrow \beta) \in \Pi$ *and* $\mathit{sup}_{\mathbf{A}}(\Pi, S, S) \neq \emptyset$ *or* $\mathbf{A}^{\boldsymbol{T}} \cap S = \emptyset$ *for every* $S \subseteq \mathit{atom}(\Pi)$.

Proof. By Lemma B.13, for every literal $l = v$ or $l = \mathit{not}\ v$, where $v \in \mathit{atom}(\Pi) \cup \mathit{conj}(\Pi) \cup \mathit{card}(\Pi) \cup \mathit{disj}(\Pi)$, we have that $\boldsymbol{t}l \in \mathbf{A}$ iff $X \models \tau[l]$, and that $\boldsymbol{f}l \in \mathbf{A}$ iff $X \not\models \tau[l]$. Hence, we can treat such conditions as synonyms in the following consideration of the implications of the statement:

(⇒) Assume that X is an answer set of Π. Then, for every $(\alpha \leftarrow \beta) \in \Pi$, we have that $X \models \big(\tau[\beta] \rightarrow \tau[\alpha]\big)$ if $\alpha \notin \mathit{card}(\Pi)$, and that $X \models \big(\tau[\beta] \rightarrow \big(\tau[\alpha] \wedge \bigwedge_{p \in \mathit{atom}(\alpha)}(p \vee \neg p)\big)\big)$ if $\alpha \in \mathit{card}(\Pi)$. This implies that $X \not\models \tau[\beta]$ or $X \models \tau[\alpha]$, from which we conclude that $\boldsymbol{t}\beta \notin \mathbf{A}$ or $\boldsymbol{f}\alpha \notin \mathbf{A}$. Furthermore, for any $S \subseteq \mathit{atom}(\Pi)$ such that $\mathbf{A}^{\boldsymbol{T}} \cap S = X \cap S \neq \emptyset$, we have that $Y = X \setminus S \subset X$ is not a model of $(\tau[\Pi])^X$. That is, $Y \not\models \phi^X$ for some $\phi \in \tau[\Pi]$, where $\phi = \big(\tau[\beta] \rightarrow \tau[\alpha]\big)$ if $\alpha \notin \mathit{card}(\Pi)$ or $\phi = \big(\tau[\beta] \rightarrow \big(\tau[\alpha] \wedge \bigwedge_{p \in \mathit{atom}(\alpha)}(p \vee \neg p)\big)\big)$ if $\alpha \in \mathit{card}(\Pi)$ for some $(\alpha \leftarrow \beta) \in \Pi$. In view of $X \models \phi$ but $Y \not\models \phi^X$, we conclude that $\phi^X \neq \bot$, $Y \models (\tau[\beta])^X$, $X \models \tau[\beta]$, and $X \models \tau[\alpha]$. Furthermore, from $X \models \tau[\beta]$, we immediately obtain $\boldsymbol{f}\beta \notin \mathbf{A}$.

Given $Y \models (\tau[\beta])^X$, we first show that $\overrightarrow{\mathit{sup}}_{\mathbf{A}}(\beta, S)$ holds. The following cases are possible:

1. $\beta = not\ v$ for some $v \in atom(\Pi) \cup conj(\Pi) \cup card(\Pi)$, so that $\overrightarrow{sup}_{\mathbf{A}}(\beta, S)$ holds.

2. $\beta \in Y = X \setminus S$, so that $\beta \in atom(\Pi) \setminus S$ and $\overrightarrow{sup}_{\mathbf{A}}(\beta, S)$ hold.

3. $\beta = j\{l_1, \ldots, l_n\}k \in card(\Pi)$ and

$$(\tau[\beta])^X = \Big(\bigwedge_{L \subseteq \{l_1,\ldots,l_n\}, |L|<j \text{ or } k<|L|} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L} \tau[l])\big)\Big)^X.$$

   Since $Y \models (\tau[\beta])^X$, for any $L \subseteq \{l_1, \ldots, l_n\}$ such that $|L| < j$, we have that $\{l \in L \mid \boldsymbol{f}l \in \mathbf{A}\} \neq \emptyset$, $L \cap S \neq \emptyset$, or $\{l \in \{l_1, \ldots, l_n\} \setminus L \mid \boldsymbol{f}l \notin \mathbf{A}\} \not\subseteq S$. However, regarding $L' = \{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{f}l \notin \mathbf{A}\}$, it holds that $\{l \in L' \mid \boldsymbol{f}l \in \mathbf{A}\} = \emptyset$, $L' \cap S = \emptyset$, and $\{l \in \{l_1, \ldots, l_n\} \setminus L' \mid \boldsymbol{f}l \notin \mathbf{A}\} \subseteq S$. It follows that $|L'| \geq j$, so that $\overrightarrow{sup}_{\mathbf{A}}(\beta, S)$ holds.

4. $\beta = \{l_1, \ldots, l_n\} \in conj(\Pi)$ and $(\tau[\beta])^X = (\bigwedge_{l \in \{l_1,\ldots,l_n\}} \tau[l])^X = \bigwedge_{l \in \{l_1,\ldots,l_n\}} (\tau[l])^X$. Since $Y \models (\tau[\beta])^X$, we conclude that $Y \models (\tau[l])^X$ for every $l \in \{l_1, \ldots, l_n\}$. Given this, one of the first three cases applies to each $l \in \{l_1, \ldots, l_n\}$, from which we conclude that $\overrightarrow{sup}_{\mathbf{A}}(l, S)$ holds, so that $\overrightarrow{sup}_{\mathbf{A}}(\beta, S)$ holds as well.

We have thus shown that $\overrightarrow{sup}_{\mathbf{A}}(\beta, S)$ holds.

We now turn to proving that $\overleftarrow{sup}_{\mathbf{A}}(\alpha, S)$ holds. For this, note that, if $\alpha \notin card(\Pi)$, $X \models \tau[\alpha]$ but $Y \not\models (\tau[\alpha])^X$ yield $\alpha \in atom(\Pi) \cup disj(\Pi)$. Hence, the following cases are possible:

1. $\alpha \in S$, so that $\overleftarrow{sup}_{\mathbf{A}}(\alpha, S)$ holds.

2. $\alpha = \{l_1; \ldots; l_n\} \in disj(\Pi)$ and $\emptyset \neq \{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\} \subseteq S$. That is, $\{l_1, \ldots, l_n\} \cap S \neq \emptyset$ and $\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\} = \emptyset$, so that $\overleftarrow{sup}_{\mathbf{A}}(\alpha, S)$ holds.

3. $\alpha = j\{l_1, \ldots, l_n\}k \in card(\Pi)$ and $(\{l_1, \ldots, l_n\} \cap X) \cap S \neq \emptyset$ because $X \models \tau[\alpha]$ but $Y \not\models \big(\tau[\alpha] \wedge \bigwedge_{p \in atom(\alpha)} (p \vee \neg p)\big)^X$.[3] Furthermore, $X \models \tau[\alpha]$ implies $|\{l \in \{l_1, \ldots, l_n\} \mid \boldsymbol{t}l \in \mathbf{A}\}| \leq k$. Along with $(\{l_1, \ldots, l_n\} \cap X) \cap S \neq \emptyset$, that is, $\{l \in \{l_1, \ldots, l_n\} \cap S \mid \boldsymbol{t}l \in \mathbf{A}\} \neq \emptyset$, we conclude that $|\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\}| < k$, so that $\overleftarrow{sup}_{\mathbf{A}}(\alpha, S)$ holds.

We have thus shown that $\overleftarrow{sup}_{\mathbf{A}}(\alpha, S)$ holds. Along with the previous observations that $\boldsymbol{f}\beta \notin \mathbf{A}$ and that $\overrightarrow{sup}_{\mathbf{A}}(\beta, S)$ holds, we conclude that $(\alpha \leftarrow \beta) \in sup_{\mathbf{A}}(\Pi, S, S)$, so that $sup_{\mathbf{A}}(\Pi, S, S) \neq \emptyset$. Since the choice of $S \subseteq atom(\Pi)$ such that $\mathbf{A}^{\boldsymbol{T}} \cap S \neq \emptyset$ was arbitrary, this establishes that $sup_{\mathbf{A}}(\Pi, S, S) \neq \emptyset$ or $\mathbf{A}^{\boldsymbol{T}} \cap S = \emptyset$ for every $S \subseteq atom(\Pi)$.

($\Leftarrow$) Assume that X is not an answer set of Π. Then, there is either some $(\alpha \leftarrow \beta) \in \Pi$ such that $X \models \tau[\beta]$ and $X \not\models \tau[\alpha]$ or some $Y \subset X$ such that $Y \models (\tau[\Pi])^X$. In the former case, we have that $\boldsymbol{t}\beta \in \mathbf{A}$ and $\boldsymbol{f}\alpha \in \mathbf{A}$ for some $(\alpha \leftarrow \beta) \in \Pi$. In the latter case, let $S = X \setminus Y$. Then, it holds that $\emptyset \neq \mathbf{A}^{\boldsymbol{T}} \cap S = S$. For the sake of contradiction, assume that $sup_{\mathbf{A}}(\Pi, S, S) \neq \emptyset$, that is, $(\alpha \leftarrow \beta) \in \Pi$ such that $\boldsymbol{f}\beta \notin \mathbf{A}$, $\overleftarrow{sup}_{\mathbf{A}}(\alpha, S)$, and $\overrightarrow{sup}_{\mathbf{A}}(\beta, S)$ hold.

In view of $\overleftarrow{sup}_{\mathbf{A}}(\alpha, S)$, the following cases are possible:

[3] Note that all atoms occurring in $\big(\tau[\alpha] \wedge \bigwedge_{p \in atom(\alpha)} (p \vee \neg p)\big)^X$ belong to $\{l_1, \ldots, l_n\} \cap X$.

1. $\alpha \in S$, $(\tau[\alpha])^X = \alpha$, and so

$$Y \not\models (\tau[\alpha])^X.$$

2. $\alpha = \{l_1; \ldots; l_n\} \in \mathit{disj}(\Pi)$, $\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\} = \emptyset$, $(\tau[\alpha])^X \equiv \bigvee_{l \in \{l_1,\ldots,l_n\} \cap S} \tau[l] = \bigvee_{p \in \{l_1,\ldots,l_n\} \cap S} p$, and so

$$Y \not\models (\tau[\alpha])^X.$$

3. $\alpha = j\{l_1, \ldots, l_n\}k \in \mathit{card}(\Pi)$, $\{l_1, \ldots, l_n\} \cap S = \mathit{atom}(\alpha) \cap S \neq \emptyset$, and so

$$Y \not\models \big(\tau[\alpha] \wedge \textstyle\bigwedge_{p \in \mathit{atom}(\alpha)} (p \vee \neg p)\big)^X.$$

We have thus shown that $Y \not\models (\tau[\alpha])^X$ if $\alpha \notin \mathit{card}(\Pi)$, and that $Y \not\models \big(\tau[\alpha] \wedge \bigwedge_{p \in \mathit{atom}(\alpha)} (p \vee \neg p)\big)^X$ if $\alpha \in \mathit{card}(\Pi)$.

We now turn to β, for which $\boldsymbol{f}\beta \notin \mathbf{A}$ implies $\boldsymbol{t}\beta \in \mathbf{A}$, that is, $X \models \tau[\beta]$. Furthermore, we have that $\overrightarrow{\mathit{sup}}_{\mathbf{A}}(\beta, S)$ holds, and the following cases are possible:

1. $\beta = \mathit{not}\ v$ for some $v \in \mathit{atom}(\Pi) \cup \mathit{conj}(\Pi) \cup \mathit{card}(\Pi)$, $(\tau[\beta])^X = \neg\bot$, and so

$$Y \models (\tau[\beta])^X.$$

2. $\beta \in \mathit{atom}(\Pi) \setminus S$, $(\tau[\beta])^X = \beta \in Y$, and so

$$Y \models (\tau[\beta])^X.$$

3. $\beta = j\{l_1, \ldots, l_n\}k \in \mathit{card}(\Pi)$ and

$$(\tau[\beta])^X = \Big(\textstyle\bigwedge_{L \subseteq \{l_1,\ldots,l_n\}, |L| < j \text{ or } k < |L|} \big((\bigwedge_{l \in L} \tau[l]) \rightarrow (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L} \tau[l])\big)\Big)^X.$$

   For any $L \subseteq \{l_1, \ldots, l_n\}$ such that $k < |L|$, $X \models \tau[\beta]$ implies $(\bigwedge_{l \in L} \tau[l])^X = \bot$, so that $Y \not\models (\bigwedge_{l \in L} \tau[l])^X$. Furthermore, since $\overrightarrow{\mathit{sup}}_{\mathbf{A}}(\beta, S)$ holds, we have that $|\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{f}l \notin \mathbf{A}\}| \geq j$. Hence, for any $L \subseteq \{l_1, \ldots, l_n\}$ such that $|L| < j$, it holds that $\{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{f}l \notin \mathbf{A}\} = \{l \in \{l_1, \ldots, l_n\} \setminus S \mid \boldsymbol{t}l \in \mathbf{A}\} \not\subseteq L$ and $\{l \in \{l_1, \ldots, l_n\} \setminus L \mid \boldsymbol{t}l \in \mathbf{A}\} \not\subseteq S$, so that $Y \models (\bigvee_{l \in \{l_1,\ldots,l_n\} \setminus L} \tau[l])^X$. Combining the cases for $|L| < j$ and $k < |L|$ yields that

$$Y \models (\tau[\beta])^X.$$

4. $\beta = \{l_1, \ldots, l_n\} \in \mathit{conj}(\Pi)$ and $(\tau[\beta])^X = (\bigwedge_{l \in \{l_1,\ldots,l_n\}} \tau[l])^X = \bigwedge_{l \in \{l_1,\ldots,l_n\}} (\tau[l])^X$. For every $l \in \{l_1, \ldots, l_n\}$, $X \models \tau[\beta]$ and $\overrightarrow{\mathit{sup}}_{\mathbf{A}}(\beta, S)$ imply $X \models \tau[l]$ and $\overrightarrow{\mathit{sup}}_{\mathbf{A}}(l, S)$. Given this, one of the first three cases applies to each $l \in \{l_1, \ldots, l_n\}$, from which we conclude that $Y \models (\tau[l])^X$, and so

$$Y \models (\tau[\beta])^X.$$

We have thus shown that $Y \models (\tau[\beta])^X$. Along with $Y \not\models (\tau[\alpha])^X$ if $\alpha \notin \mathit{card}(\Pi)$ and $Y \not\models \big(\tau[\alpha] \wedge \bigwedge_{p \in \mathit{atom}(\alpha)}(p \vee \neg p)\big)^X$ if $\alpha \in \mathit{card}(\Pi)$, we further conclude that $Y \not\models \big(\tau[\beta] \rightarrow \tau[\alpha]\big)^X$ if $\alpha \notin \mathit{card}(\Pi)$ and $Y \not\models \big(\tau[\beta] \rightarrow \big(\tau[\alpha] \wedge \bigwedge_{p \in \mathit{atom}(\alpha)}(p \vee \neg p)\big)\big)^X$ if $\alpha \in \mathit{card}(\Pi)$. That is, $Y \not\models (\tau[\Pi])^X$, which is a contradiction to our initial assumption. This shows that $\mathit{sup}_{\mathbf{A}}(\Pi, S, S) \neq \emptyset$ cannot be the case, so that $\mathit{sup}_{\mathbf{A}}(\Pi, S, S) = \emptyset$ must hold. In addition, $\emptyset \neq \mathbf{A}^{\boldsymbol{T}} \cap S = S$ holds by the choice of $S = X \setminus Y$. □

We are now ready to show Theorem 3.9, 3.10, 3.13, and 3.14, stating the soundness and completeness of tableau calculi for unary, conjunctive, cardinality, and disjunctive programs, respectively. Since disjunctive programs include unary, conjunctive, and cardinality programs, it is sufficient to prove Theorem 3.14.

Theorem 3.14. *Let Π be a disjunctive program.*

Then, we have that the following holds for the tableau calculus consisting of the tableau rules (a)–(v):

1. *Every incomplete tableau for Π and $\emptyset$ can be extended to a complete tableau for Π and $\emptyset$.*
2. *Program Π has an answer set X iff every complete tableau for Π and $\emptyset$ has a unique non-contradictory branch $(\Pi, \mathbf{A})$ such that $\mathbf{A}^{\boldsymbol{T}} \cap \mathit{atom}(\Pi) = X$.*
3. *Program Π has no answer set iff every complete tableau for Π and $\emptyset$ is a refutation.*

Proof. We separately consider the items of the statement:

1. By applying *Cut*$[\mathit{atom}(\Pi) \cup \mathit{conj}(\Pi) \cup \mathit{card}(\Pi) \cup \mathit{disj}(\Pi)]$, an incomplete branch in a tableau for Π and $\emptyset$ can be extended to a subtableau such that, for every branch $(\Pi, \mathbf{A})$ in it, we have that $\mathit{atom}(\Pi) \cup \mathit{conj}(\Pi) \cup \mathit{card}(\Pi) \cup \mathit{disj}(\Pi) \subseteq \mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}}$. Furthermore, if $(\Pi, \mathbf{A})$ is not complete, then $D_{\{\text{(a)–(f),(h)–(v)}\}}(\Pi, \mathbf{A}) \not\subseteq \mathbf{A}$, so that the application of some of the tableau rules (a)–(f) in Figure 3.4 or (h)–(v) in Figure 3.5, 3.7, and 3.8 yields a contradictory and thus complete branch.
2. By Theorem B.15, for every $X \subseteq \mathit{atom}(\Pi)$, we have that X is an answer set of Π iff the total assignment

$$\begin{aligned} \mathbf{A} \;=\; & \{\boldsymbol{T}v \mid v \in \mathit{atom}(\Pi) \cup \mathit{conj}(\Pi) \cup \mathit{card}(\Pi) \cup \mathit{disj}(\Pi), X \models \tau[v]\} \\ \cup\; & \{\boldsymbol{F}v \mid v \in \mathit{atom}(\Pi) \cup \mathit{conj}(\Pi) \cup \mathit{card}(\Pi) \cup \mathit{disj}(\Pi), X \not\models \tau[v]\} \end{aligned}$$

   is such that $\boldsymbol{t}\beta \notin \mathbf{A}$ or $\boldsymbol{f}\alpha \notin \mathbf{A}$ for every $(\alpha \leftarrow \beta) \in \Pi$ and $\mathit{sup}_{\mathbf{A}}(\Pi, S, S) \neq \emptyset$ or $\mathbf{A}^{\boldsymbol{T}} \cap S = \emptyset$ for every $S \subseteq \mathit{atom}(\Pi)$. Given this, we separately show the implications of the second item:

   (⇒) Assume that X is an answer set of Π. Then, Lemma B.10, B.12, and B.14 establish that $D_{\{\text{(a)–(f),(h)–(v)}\}}(\Pi, \mathbf{A}') \subseteq \mathbf{A}$ for every $\mathbf{A}' \subseteq \mathbf{A}$. Furthermore, for any application of *Cut*$[\mathit{atom}(\Pi) \cup \mathit{conj}(\Pi) \cup \mathit{card}(\Pi) \cup \mathit{disj}(\Pi)]$ on a branch $(\Pi, \mathbf{A}')$ such that $\mathbf{A}' \subseteq \mathbf{A}$, we have that the assignment in exactly one of the resulting branches is contained in $\mathbf{A}$. Along with $\emptyset \subseteq \mathbf{A}$, it follows that every complete tableau for Π and $\emptyset$ has a non-contradictory branch $(\Pi, \mathbf{A})$ such that $\mathbf{A}^{\boldsymbol{T}} \cap \mathit{atom}(\Pi) = X$. By Lemma B.7, B.8, and B.9, we also have that $(\Pi, \mathbf{A})$ is the unique non-contradictory complete branch such that $\mathbf{A}^{\boldsymbol{T}} \cap \mathit{atom}(\Pi) = X$.

($\Leftarrow$) Assume that $(\Pi, \mathbf{A})$ is a non-contradictory complete branch. Then, for every $v \in atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \cup disj(\Pi)$, Lemma B.7, B.8, and B.9 establish that $\boldsymbol{T}v \in \mathbf{A}$ iff $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) \models \tau[v]$. Furthermore, Lemma B.4 and B.5 show that $\boldsymbol{t}\beta \notin \mathbf{A}$ or $\boldsymbol{f}\alpha \notin \mathbf{A}$ for every $(\alpha \leftarrow \beta) \in \Pi$ and that $sup_{\mathbf{A}}(\Pi, S, S) \neq \emptyset$ or $\mathbf{A}^{\boldsymbol{T}} \cap S = \emptyset$ for every $S \subseteq atom(\Pi)$. By Theorem B.15, we conclude that $X = \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is an answer set of Π.

3. From the second item, if Π has an answer set, then every complete tableau for Π and $\emptyset$ has a non-contradictory branch; by the first item, there is some complete tableau for Π and $\emptyset$, so that some complete tableau for Π and $\emptyset$ is not a refutation. Conversely, if some complete tableau for Π and $\emptyset$ is not a refutation, it has a non-contradictory branch $(\Pi, \mathbf{A})$, and $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is an answer set of Π, as shown in the proof of the second item.

We have thus shown that all items of the statement hold. □

Proofs of Correspondences on Normal Programs

We now show the correspondences stated in Proposition 3.11 and 3.12 between the basic tableau rules in Figure 3.1 on Page 19 and the (generic) tableau rules in Figure 3.4 and 3.5 on Page 34 and 37, respectively, on the common class of normal programs.

Proposition 3.11. *Let Π be a normal program, $\mathbf{A}$ an assignment, and F, G any pair of a basic tableau rule F and a generic tableau rule G belonging to the same line in Table 3.1.*

Then, we have that

1. $D_{\{F\}}(\Pi, \mathbf{A}) = D_{\{G\}}(\Pi, \mathbf{A})$ *if* $F \notin \{$*BTA*, *WFJ*$[2^{atom(\Pi)}]\}$;
2. $D_{\{$*BTA*$\}}(\Pi, \mathbf{A}) \supseteq D_{\{N\downarrow\}}(\Pi, \mathbf{A})$ *and, if* $D_{\{$*BTA*$\}}(\Pi, \mathbf{A}) \neq D_{\{N\downarrow\}}(\Pi, \mathbf{A})$, *then* $\mathbf{A} \cup D_{\{N\uparrow\}}(\Pi, \mathbf{A})$ *is contradictory;*
3. $D_{\{$*WFJ*$[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) \supseteq D_{\{U\downarrow\}}(\Pi, \mathbf{A})$ *and, if* $\boldsymbol{T}B \in D_{\{$*WFJ*$[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) \setminus D_{\{U\downarrow\}}(\Pi, \mathbf{A})$, *then* $\mathbf{A} \cup D_{\{U\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{F}B\})$ *is contradictory.*

Proof. The correspondences are obvious for the pairs *(c)*, *(a)*, *(d)*, *(b)*, *(a)*, *(h)*, *(b)*, *(i)*, *(e)*, *(j)*, and *(f)*, *(k)*. It remains to show the statement for the pairs *FFA*, $N\uparrow$, *BTA*, $N\downarrow$, *WFN*$[2^{atom(\Pi)}]$, $U\uparrow$, and *WFJ*$[2^{atom(\Pi)}]$, $U\downarrow$:

(*FFA*, $N\uparrow$) We have that $\boldsymbol{F}p \in D_{\{$*FFA*$\}}(\Pi, \mathbf{A})$ iff $p \in atom(\Pi)$ such that $body_{\Pi}(p) \subseteq \mathbf{A}^{\boldsymbol{F}}$ iff $p \in atom(\Pi)$ such that $sup_{\mathbf{A}}(\Pi, \{p\}, \emptyset) = \emptyset$ iff $\boldsymbol{F}p \in D_{\{N\uparrow\}}(\Pi, \mathbf{A})$.

(*BTA*, $N\downarrow$) If $\boldsymbol{T}B \in D_{\{N\downarrow\}}(\Pi, \mathbf{A})$, then $sup_{\mathbf{A}}(\Pi, \{p\}, \emptyset) = \{p \leftarrow B\}$ for some $p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$, so that $\alpha \neq p$ or $\boldsymbol{F}\beta \in \mathbf{A}$ for every $(\alpha \leftarrow \beta) \in \Pi \setminus \{p \leftarrow B\}$. From this, we conclude that $body_{\Pi}(p) \setminus \mathbf{A}^{\boldsymbol{F}} = \{B\}$, so that $\boldsymbol{T}B \in D_{\{$*BTA*$\}}(\Pi, \mathbf{A})$. Furthermore, if $\boldsymbol{T}B' \in D_{\{$*BTA*$\}}(\Pi, \mathbf{A}) \setminus D_{\{N\downarrow\}}(\Pi, \mathbf{A})$, then $body_{\Pi}(p') \setminus \mathbf{A}^{\boldsymbol{F}} \subseteq \{B'\}$ for some $B' \in body(\Pi)$ and $p' \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$, which implies that $sup_{\mathbf{A}}(\Pi, \{p'\}, \emptyset) \subseteq \{p' \leftarrow B'\}$. However, $\boldsymbol{T}B' \notin D_{\{N\downarrow\}}(\Pi, \mathbf{A})$ yields that $(p' \leftarrow B') \notin sup_{\mathbf{A}}(\Pi, \{p'\}, \emptyset)$. Hence, we have that $sup_{\mathbf{A}}(\Pi, \{p'\}, \emptyset) = \emptyset$, and $\mathbf{A} \cup D_{\{N\uparrow\}}(\Pi, \mathbf{A})$ is contradictory because $p' \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$.

(*WFN*$[2^{atom(\Pi)}]$, $U\uparrow$) We have that $\boldsymbol{F}p \in D_{\{\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A})$ iff $p \in S$ for some $S \subseteq atom(\Pi)$ such that $EB_\Pi(S) \subseteq \mathbf{A}^{\boldsymbol{F}}$ iff $p \in S$ for some $S \subseteq atom(\Pi)$ such that $sup_{\mathbf{A}}(\Pi, S, S) = \emptyset$ iff $\boldsymbol{F}p \in D_{\{U\uparrow\}}(\Pi, \mathbf{A})$.

(*WFJ*$[2^{atom(\Pi)}]$, $U\downarrow$) If $\boldsymbol{T}B \in D_{\{U\downarrow\}}(\Pi, \mathbf{A})$, then $sup_{\mathbf{A}}(\Pi, S, S) = \{p \leftarrow B\}$, where $p \in S$ for some $S \subseteq atom(\Pi)$ such that $\mathbf{A}^{\boldsymbol{T}} \cap S \neq \emptyset$. From this, we conclude that $EB_\Pi(S) \setminus \mathbf{A}^{\boldsymbol{F}} = \{B\}$, so that $\boldsymbol{T}B \in D_{\{\mathit{WFJ}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A})$. Furthermore, if $\boldsymbol{T}B' \in D_{\{\mathit{WFJ}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}) \setminus D_{\{U\downarrow\}}(\Pi, \mathbf{A})$, then $EB_\Pi(S') \setminus \mathbf{A}^{\boldsymbol{F}} \subseteq \{B'\}$ for some $B' \in body(\Pi)$ and $S' \subseteq atom(\Pi)$ such that $\mathbf{A}^{\boldsymbol{T}} \cap S' \neq \emptyset$, which implies that $sup_{\mathbf{A}}(\Pi, S', S') \subseteq \{p' \leftarrow B' \mid p' \in S'\}$. In view of Lemma B.10 and B.11, we have that $sup_{\mathbf{A} \cup \{\boldsymbol{F}B'\}}(\Pi, S', S') = \emptyset$, and $\mathbf{A} \cup D_{\{U\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{F}B'\})$ is contradictory because $\mathbf{A}^{\boldsymbol{T}} \cap S' \neq \emptyset$.

We have thus shown that the stated correspondences according to Table 3.1 hold. □

Proposition 3.12. *Let* Π *be a normal program,* $\mathbf{A}$ *an assignment,* $\mathcal{T}$ *a tableau calculus containing any subset of the tableau rules in Figure 3.1 for* $\Omega = 2^{atom(\Pi)}$*, and* $\mathcal{T}'$ *the generic image of* $\mathcal{T}$*.*

If ***FFA*** $\in \mathcal{T}$ *or* ***BTA*** $\notin \mathcal{T}$ *and if* ***WFJ***$[\Omega] \in \mathcal{T}$ *implies that* $\{$***FTB***, ***FFB***, ***WFN***$[\Omega]$, ***Cut***$[\Gamma]\} \subseteq \mathcal{T}$ *for* $\Gamma \subseteq atom(\Pi) \cup body(\Pi)$ *such that* $atom(\Pi) \subseteq \Gamma$ *or* $body(\Pi) \subseteq \Gamma$*, then we have that the following holds:*

1. *For every complete tableau of* $\mathcal{T}$ *for* Π *and* $\mathbf{A}$ *with* n *branches, there is a complete tableau of* $\mathcal{T}'$ *for* Π *and* $\mathbf{A}$ *with the same non-contradictory branches and at most* $(\max\{|atom(\Pi)|, |body(\Pi)|\} + 1) * n$ *branches overall.*

2. *Every tableau of* $\mathcal{T}'$ *for* Π *and* $\mathbf{A}$ *is a tableau of* $\mathcal{T}$ *for* Π *and* $\mathbf{A}$*.*

Proof. Assume that ***FFA*** $\in \mathcal{T}$ or ***BTA*** $\notin \mathcal{T}$ and that ***WFJ***$[\Omega] \in \mathcal{T}$ implies that $\{$***FTB***, ***FFB***, ***WFN***$[\Omega]$, ***Cut***$[\Gamma]\} \subseteq \mathcal{T}$ for $\Gamma \subseteq atom(\Pi) \cup body(\Pi)$ such that $atom(\Pi) \subseteq \Gamma$ or $body(\Pi) \subseteq \Gamma$. By Proposition 3.11, we immediately conclude that every tableau of $\mathcal{T}'$ for Π and $\mathbf{A}$ is a tableau of $\mathcal{T}$ for Π and $\mathbf{A}$ as well. Furthermore, in view of the first two items in the statement of Proposition 3.11, we have that any application of a tableau rule in $\mathcal{T}$ other than ***WFJ***$[\Omega]$ on a branch $(\Pi, \mathbf{A}')$ extending $(\Pi, \mathbf{A})$ leads to the same result, in terms of deduced entries or a contradiction, respectively, by applying a corresponding tableau rule in $\mathcal{T}'$. Hence, it is sufficient to show that, if there is some $\boldsymbol{T}B \in D_{\{\mathit{WFJ}[\Omega]\}}(\Pi, \mathbf{A}') \setminus (\mathbf{A}' \cup D_{\{TC\uparrow, U\downarrow\}}(\Pi, \mathbf{A}'))$, there is a corresponding subtableau of $\mathcal{T}'$ that introduces at most $|(B^+ \cup B^-) \setminus ((\mathbf{A}')^{\boldsymbol{T}} \cup (\mathbf{A}')^{\boldsymbol{F}})|$ contradictory branches, while a single remaining branch includes $\boldsymbol{T}B$ (and possibly further entries belonging to any non-contradictory branch extending $(\Pi, \mathbf{A}' \cup \{\boldsymbol{T}B\})$ in a complete tableau of $\mathcal{T}$ for Π and $\mathbf{A}$). To this end, assume that $\boldsymbol{T}B \in D_{\{\mathit{WFJ}[\Omega]\}}(\Pi, \mathbf{A}') \setminus (\mathbf{A}' \cup D_{\{TC\uparrow, U\downarrow\}}(\Pi, \mathbf{A}'))$. Then, $EB_\Pi(S) \setminus (\mathbf{A}')^{\boldsymbol{F}} \subseteq \{B\}$ for some $S \subseteq atom(\Pi)$ such that $(\mathbf{A}')^{\boldsymbol{T}} \cap S \neq \emptyset$, $sup_{\mathbf{A}'}(\Pi, S, S) \subseteq \{p \leftarrow B \mid p \in S\}$, and $|sup_{\mathbf{A}'}(\Pi, S, S)| \neq 1$. Furthermore, one of the following cases applies:

($sup_{\mathbf{A}'}(\Pi, S, S) = \emptyset$) We have that $\boldsymbol{F}p \in D_{\{U\uparrow\}}(\Pi, \mathbf{A}')$ for every $p \in S$. Given that $(\mathbf{A}')^{\boldsymbol{T}} \cap S \neq \emptyset$, we conclude that $(\Pi, \mathbf{A}')$ can be extended to a contradictory branch by an application of $U\uparrow$.

($sup_{\mathbf{A}'}(\Pi, S, S) \neq \emptyset$) In view of Lemma B.10 and B.11, we have that $sup_{\mathbf{A}' \cup \{\boldsymbol{F}B\}}(\Pi, S, S) = \emptyset$, so that an application of $U\!\uparrow$ is sufficient to contradict any extension of $(\Pi, \mathbf{A}')$ including $\boldsymbol{F}B$. In particular, if $\boldsymbol{F}B \in D_{\{FC\uparrow\}}(\Pi, \mathbf{A}')$, we can extend $(\Pi, \mathbf{A}')$ to a contradictory branch without cutting. Otherwise, if $\mathit{Cut}[\Gamma] \in \mathcal{T}$ such that $\mathit{body}(\Pi) \subseteq \Gamma$, we can cut on B, contradict the branch for $\boldsymbol{F}B$ by applying $U\!\uparrow$, and proceed with the branch $(\Pi, \mathbf{A}' \cup \{\boldsymbol{T}B\})$, also obtained by applying $\mathit{WFJ}[\Omega]$. Alternatively, if $\mathit{Cut}[\Gamma] \in \mathcal{T}$ such that $atom(\Pi) \subseteq \Gamma$, we can successively cut on atoms in $(B^+ \cup B^-) \setminus ((\mathbf{A}')^{\boldsymbol{T}} \cup (\mathbf{A}')^{\boldsymbol{F}})$ and contradict a branch for $\boldsymbol{f}l$, where $l \in B$, by applying $FC\!\uparrow$ and $U\!\uparrow$. Provided that $B^+ \cap B^- = \emptyset$,[4] this strategy yields a single branch $(\Pi, \mathbf{A}' \cup \{\boldsymbol{t}l \mid l \in B\})$, which can be further extended to $(\Pi, \mathbf{A}' \cup \{\boldsymbol{t}l \mid l \in B\} \cup \{\boldsymbol{T}B\})$ by an application of $TC\!\uparrow$. Given that $\mathit{FFB} \in \mathcal{T}$, we also have that any non-contradictory branch extending $(\Pi, \mathbf{A}' \cup \{\boldsymbol{T}B\})$ in a complete tableau of $\mathcal{T}$ for Π and $\mathbf{A}$ contains $\boldsymbol{t}l$ for all $l \in B$.

We have thus shown that an entry $\boldsymbol{T}B \in D_{\{\mathit{WFJ}[\Omega]\}}(\Pi, \mathbf{A}') \setminus (\mathbf{A}' \cup D_{\{TC\uparrow, U\downarrow\}}(\Pi, \mathbf{A}'))$ can also be generated in the single (if any) non-contradictory branch in a subtableau of $\mathcal{T}'$ extending $(\Pi, \mathbf{A}')$ and admitting the same non-contradictory extensions as $(\Pi, \mathbf{A}' \cup \{\boldsymbol{T}B\})$ in a complete tableau of $\mathcal{T}$ for Π and $\mathbf{A}$, while introducing at most $\max\{|atom(\Pi)|, |\mathit{body}(\Pi)|\}$ contradictory branches overall along each branch in a complete tableau of $\mathcal{T}$ for Π and $\mathbf{A}$. □

The previous results allow us to derive Theorem 3.1 as a consequence of Theorem 3.10 (i.e., Theorem 3.14 restricted to the class of conjunctive programs).

Theorem 3.1. *Let Π be a normal program.*

Then, we have that the following holds for tableau calculi $\mathcal{T}_{\mathit{smodels}}$, $\mathcal{T}_{\mathit{nomore}}$, and $\mathcal{T}_{\mathit{nomore}++}$:

1. *Every incomplete tableau for Π and $\emptyset$ can be extended to a complete tableau for Π and $\emptyset$.*
2. *Program Π has an answer set X iff every complete tableau for Π and $\emptyset$ has a unique non-contradictory branch $(\Pi, \mathbf{A})$ such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$.*
3. *Program Π has no answer set iff every complete tableau for Π and $\emptyset$ is a refutation.*

Proof. By Proposition 3.11, $\mathcal{T}_{\mathit{smodels}}$, $\mathcal{T}_{\mathit{nomore}}$, and $\mathcal{T}_{\mathit{nomore}++}$ admit the same non-contradictory complete branches as the tableau calculus consisting of the tableau rules *(a)–(k)* in Figure 3.4 and 3.5; in particular, if $\boldsymbol{T}B \in D_{\{U\downarrow\}}(\Pi, \mathbf{A})$ for a branch $(\Pi, \mathbf{A})$, we have that $\boldsymbol{T}B \in D_{\{\mathit{WFJ}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A})$, so that $\mathbf{A} \cup D_{\{\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{F}B\})$ is contradictory (cf. Figure 3.1).[5] Hence, from Theorem 3.10 and the fact that answer sets of $\tau[\Pi]$ match answer sets (as introduced in Section 2.1) of Π (cf. [162]), the result follows immediately for $\mathcal{T}_{\mathit{nomore}++}$. Moreover, for $\mathcal{T}_{\mathit{smodels}}$ and $\mathcal{T}_{\mathit{nomore}}$, using $\mathit{Cut}[atom(\Pi)]$

[4] If $B^+ \cap B^- \neq \emptyset$, all branches in a subtableau of $\mathcal{T}'$ obtained by successively cutting on atoms in $(B^+ \cup B^-) \setminus ((\mathbf{A}')^{\boldsymbol{T}} \cup (\mathbf{A}')^{\boldsymbol{F}})$ and contradicting branches for $\boldsymbol{f}l$, where $l \in B$, are contradictory. Given that $\mathit{FFB} \in \mathcal{T}$, any branch extending $(\Pi, \mathbf{A}' \cup \{\boldsymbol{T}B\})$ in a complete tableau of $\mathcal{T}$ for Π and $\mathbf{A}$ is contradictory too.

[5] Every non-contradictory complete branch has exactly one occurrence in any complete tableau of the tableau calculus containing *(a)–(k)*, $\mathcal{T}_{\mathit{smodels}}$, $\mathcal{T}_{\mathit{nomore}}$, or $\mathcal{T}_{\mathit{nomore}++}$ for Π and $\emptyset$. For the former, this is established by Lemma B.4, B.5, B.8, B.10, B.12, and B.14 (along with the fact that *Cut* applications preserve non-contradictory complete branches). For $\mathcal{T}_{\mathit{smodels}}$, $\mathcal{T}_{\mathit{nomore}}$, and $\mathcal{T}_{\mathit{nomore}++}$, it follows from the observation that $D_{\{\mathit{(a)–(h)},\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A}') \subseteq D_{\{\mathit{(a)–(h)},\mathit{WFN}[2^{atom(\Pi)}]\}}(\Pi, \mathbf{A})$ for every assignment $\mathbf{A}$ and every $\mathbf{A}' \subseteq \mathbf{A}$.

and $\mathit{Cut}[\mathit{body}(\Pi)]$, respectively, in place of $\mathit{Cut}[\mathit{atom}(\Pi) \cup \mathit{body}(\Pi)]$, it is sufficient to show that the first item of the statement holds. Regarding $\mathcal{T}_{\mathit{smodels}}$, note that, for every $B \in \mathit{body}(\Pi)$, either $\boldsymbol{T}B \in D_{\{\mathit{FTB}\}}(\Pi, \mathbf{A})$ or $\boldsymbol{F}B \in D_{\{\mathit{FFB}\}}(\Pi, \mathbf{A})$ for any non-contradictory assignment $\mathbf{A}$ such that $\mathit{atom}(\Pi) \subseteq \mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}}$, so that the first item of the statement holds for $\mathcal{T}_{\mathit{smodels}}$. Regarding $\mathcal{T}_{\mathit{nomore}}$, for every $p \in \mathit{atom}(\Pi)$, either $\boldsymbol{T}p \in D_{\{\mathit{FTA}\}}(\Pi, \mathbf{A})$ or $\boldsymbol{F}p \in D_{\{\mathit{FFA}\}}(\Pi, \mathbf{A})$ for any non-contradictory assignment $\mathbf{A}$ such that $\mathit{body}(\Pi) \subseteq \mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}}$, so that the first item of the statement holds for $\mathcal{T}_{\mathit{nomore}}$ as well. □

Along with Lemma B.3 on different variants of tableau rule *WFN*, Theorem 3.1 yields Theorem 3.8.

Theorem 3.8. *Let Π be a normal program.*
Then, we have that the following holds for tableau calculus $\mathcal{T}_{\mathit{comp}} \cup \{\mathit{WFN}[\mathit{loop}(\Pi)]\}$:

1. *Every incomplete tableau for Π and $\emptyset$ can be extended to a complete tableau for Π and $\emptyset$.*
2. *Program Π has an answer set X iff every complete tableau for Π and $\emptyset$ has a unique non-contradictory branch $(\Pi, \mathbf{A})$ such that $\mathbf{A}^{\boldsymbol{T}} \cap \mathit{atom}(\Pi) = X$.*
3. *Program Π has no answer set iff every complete tableau for Π and $\emptyset$ is a refutation.*

Proof. By Lemma B.3, $\mathcal{T}_{\mathit{nomore}++}$ and $\mathcal{T}_{\mathit{comp}} \cup \{\mathit{WFN}[\mathit{loop}(\Pi)]\}$ admit the same non-contradictory complete branches. Hence, the result follows immediately from Theorem 3.1. □

We have thus proven the formal results presented in Section 3.3, and also demonstrated Theorem 3.1 and 3.8.

B.2.3 Section 3.4

We below consider minimal refutations of tableau calculi $\mathcal{T}_{\mathit{nomore}}$, $\mathcal{T}_{\mathit{smodels}}$, $\mathcal{T}_{\mathit{card}}$, and $\mathcal{T}_{\mathit{conj}}$ for particular families of logic programs, thus showing exponential separations between $\mathcal{T}_{\mathit{nomore}}$ and $\mathcal{T}_{\mathit{smodels}}$ as well as between $\mathcal{T}_{\mathit{card}}$ and $\mathcal{T}_{\mathit{conj}}$.

Proposition 3.15. *There is an infinite family $\{\Pi^n\}$ of normal programs such that*

1. *the size of minimal refutations of $\mathcal{T}_{\mathit{nomore}}$ for Π^n is asymptotically linear in n;*
2. *the size of minimal refutations of $\mathcal{T}_{\mathit{smodels}}$ for Π^n is asymptotically exponential in n.*

Proof. Consider the following family $\{\Pi^n_a \cup \Pi^n_c\}$ of normal programs for $n \geq 1$:

$$\Pi^n_a \cup \Pi^n_c \;=\; \{x \leftarrow \mathit{not}\ x\} \cup \bigcup\nolimits_{1 \leq i \leq n} \left\{ \begin{array}{l} x \leftarrow a_i, b_i \\ a_i \leftarrow \mathit{not}\ b_i \\ b_i \leftarrow \mathit{not}\ a_i \end{array} \right\}$$

The domain of assignments $\mathbf{A}$ is $\mathit{dom}(\mathbf{A}) = \{x, \{\mathit{not}\ x\}\} \cup \bigcup_{1 \leq i \leq n} \{a_i, b_i, \{\mathit{not}\ a_i\}, \{\mathit{not}\ b_i\}, \{a_i, b_i\}\}$. We separately investigate minimal refutations of $\mathcal{T}_{\mathit{nomore}}$ and $\mathcal{T}_{\mathit{smodels}}$ for members of $\{\Pi^n_a \cup \Pi^n_c\}$:

($\mathcal{T}_{\mathit{nomore}}$) An optimal strategy to construct a refutation of $\mathcal{T}_{\mathit{nomore}}$ for $\Pi^n_a \cup \Pi^n_c$ (cf. Figure 3.11 on Page 48) is as follows:

1. Cut on $\{not\ x\}$, complete the branch for $\boldsymbol{T}\{not\ x\}$, using the deterministic tableau rules *BTB* and *FTA*, and deduce $\boldsymbol{T}x$ in the branch for $\boldsymbol{F}\{not\ x\}$, using the deterministic tableau rule *BFB*.
2. Complete the branch containing $\boldsymbol{T}x$ (and $\boldsymbol{F}\{not\ x\}$), but none of $\boldsymbol{T}\{a_i, b_i\}$ for $1 \leq i \leq n$, if it contains $n-1$ entries of the form $\boldsymbol{F}\{a_i, b_i\}$, using the deterministic tableau rules *BTA* and *BTB*. Otherwise, if there are less than $n-1$ entries of the form $\boldsymbol{F}\{a_i, b_i\}$ in the branch, cut on some unassigned $\{a_i, b_i\}$ for $1 \leq i \leq n$ and complete the branch for $\boldsymbol{T}\{a_i, b_i\}$, using the deterministic tableau rules *BTA* and *BTB*.

In a nutshell, a refutation constructed in this way makes use of immediate contradictions obtained when assigning any of the bodies $\{a_i, b_i\}$ to true, so that each application of *Cut*$[body(\Pi_a^n \cup \Pi_c^n)]$ yields one branch that is completed without cutting any further. Hence, such a refutation of $\mathcal{T}_{nomore}$ for $\Pi_a^n \cup \Pi_c^n$ is of size linear in n.

($\mathcal{T}_{smodels}$) An optimal strategy to construct a refutation of $\mathcal{T}_{smodels}$ for $\Pi_a^n \cup \Pi_c^n$ (cf. Figure 3.10 on Page 47) is as follows:

1. Cut on x, complete the branch for $\boldsymbol{F}x$, using the deterministic tableau rules *FTB* and *BFA*, and deduce $\boldsymbol{F}\{not\ x\}$ in the branch for $\boldsymbol{T}x$, using the deterministic tableau rule *FFB*.
2. Complete any of the branches containing $\boldsymbol{T}x$ (and $\boldsymbol{F}\{not\ x\}$) if the branch contains $n-1$ entries of the form $\boldsymbol{F}\{a_i, b_i\}$ for $1 \leq i \leq n$, using the deterministic tableau rules *BTA* and *BTB*. Otherwise, if there are less than $n-1$ entries of the form $\boldsymbol{F}\{a_i, b_i\}$ in a branch, cut on some unassigned a_i for $1 \leq i \leq n$ and deduce $\boldsymbol{F}\{a_i, b_i\}$ in the branch for $\boldsymbol{T}a_i$ as well as in the branch for $\boldsymbol{F}a_i$, using the deterministic tableau rules *BTA*, *BTB*, and *FFB*.

As the second step shows, cuts on atoms a_i (or b_i) for $1 \leq i \leq n$ yield symmetric alternatives, since $\boldsymbol{F}\{a_i, b_i\}$ is deduced in each of the resulting branches. That is, except for the initial cut on x, applications of *Cut*$[atom(\Pi_a^n \cup \Pi_c^n)]$ do not admit immediate contradictions and must thus be cascaded to form a perfect binary tree. Hence, a minimal refutation of $\mathcal{T}_{smodels}$ for $\Pi_a^n \cup \Pi_c^n$ is of size exponential in n.

We have thus shown that the asymptotic sizes of minimal refutations of $\mathcal{T}_{nomore}$ and $\mathcal{T}_{smodels}$ for $\Pi_a^n \cup \Pi_c^n$ are $O(n)$ and $O(2^n)$, respectively. Hence, $\mathcal{T}_{nomore}$ is not polynomially simulated by $\mathcal{T}_{smodels}$. □

Proposition 3.16. *There is an infinite family $\{\Pi^n\}$ of normal programs such that*

1. *the size of minimal refutations of $\mathcal{T}_{smodels}$ for Π^n is asymptotically linear in n;*
2. *the size of minimal refutations of $\mathcal{T}_{nomore}$ for Π^n is asymptotically exponential in n.*

Proof. Consider the following family $\{\Pi_b^n \cup \Pi_c^n\}$ of normal programs for $n \geq 1$:

$$\Pi_b^n \cup \Pi_c^n \quad = \quad \{y \leftarrow c_1, \ldots, c_n, not\ y\} \cup \bigcup\nolimits_{1 \leq i \leq n} \left\{ \begin{array}{l} c_i \leftarrow not\ a_i \\ c_i \leftarrow not\ b_i \\ a_i \leftarrow not\ b_i \\ b_i \leftarrow not\ a_i \end{array} \right\}$$

The domain of assignments $\mathbf{A}$ is $dom(\mathbf{A}) = \{y, \{c_1, \ldots, c_n, not\ y\}\} \cup \bigcup_{1 \leq i \leq n} \{a_i, b_i, c_i, \{not\ a_i\}, \{not\ b_i\}\}$. We separately investigate minimal refutations of $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ for members of $\{\Pi_b^n \cup \Pi_c^n\}$:

($\mathcal{T}_{smodels}$) An optimal strategy to construct a refutation of $\mathcal{T}_{smodels}$ for $\Pi_b^n \cup \Pi_c^n$ (cf. Figure 3.13 on Page 50) is as follows:

1. Cut on y, complete the branch for $\boldsymbol{T}y$, using the deterministic tableau rules *BTA* and *FFB*, and deduce $\boldsymbol{F}\{c_1, \ldots, c_n, not\ y\}$ in the branch for $\boldsymbol{F}y$, using the deterministic tableau rule *BFA*.
2. Complete the branch containing $\boldsymbol{F}\{c_1, \ldots, c_n, not\ y\}$ (and $\boldsymbol{F}y$), but none of $\boldsymbol{F}c_i$ for $1 \leq i \leq n$, if it contains $n-1$ entries of the form $\boldsymbol{T}c_i$, using the deterministic tableau rules *BFB*, *BFA*, and *FFA*. Otherwise, if there are less than $n-1$ entries of the form $\boldsymbol{T}c_i$ in the branch, cut on some unassigned c_i for $1 \leq i \leq n$ and complete the branch for $\boldsymbol{F}c_i$, using the deterministic tableau rules *BFB*, *BFA*, and *FFA*.

In a nutshell, a refutation constructed in this way makes use of immediate contradictions obtained when assigning any of the atoms c_i to false, so that each application of *Cut*$[atom(\Pi_b^n \cup \Pi_c^n)]$ yields one branch that is completed without cutting any further. Hence, such a refutation of $\mathcal{T}_{smodels}$ for $\Pi_b^n \cup \Pi_c^n$ is of size linear in n.

($\mathcal{T}_{nomore}$) An optimal strategy to construct a refutation of $\mathcal{T}_{nomore}$ for $\Pi_b^n \cup \Pi_c^n$ (cf. Figure 3.12 on Page 49) is as follows:

1. Cut on $\{c_1, \ldots, c_n, not\ y\}$, complete the branch for $\boldsymbol{T}\{c_1, \ldots, c_n, not\ y\}$, using the deterministic tableau rules *FTA* and *BTB*, and deduce $\boldsymbol{F}y$ in the branch for $\boldsymbol{F}\{c_1, \ldots, c_n, not\ y\}$, using the deterministic tableau rule *FFA*.
2. Complete any of the branches containing $\boldsymbol{F}\{c_1, \ldots, c_n, not\ y\}$ (and $\boldsymbol{F}y$) if the branch contains $n-1$ entries of the form $\boldsymbol{T}c_i$ for $1 \leq i \leq n$, using the deterministic tableau rules *BFB*, *BFA*, and *FFA*. Otherwise, if there are less than $n-1$ entries of the form $\boldsymbol{T}c_i$ in a branch, cut on some unassigned $\{not\ a_i\}$ for $1 \leq i \leq n$ and deduce $\boldsymbol{T}c_i$ in the branch for $\boldsymbol{T}\{not\ a_i\}$ as well as in the branch for $\boldsymbol{F}\{not\ a_i\}$, using the deterministic tableau rules *FTA*, *BFB*, and *BTA*.

As the second step shows, cuts on bodies $\{not\ a_i\}$ (or $\{not\ b_i\}$) for $1 \leq i \leq n$ yield symmetric alternatives, since $\boldsymbol{T}c_i$ is deduced in each of the resulting branches. That is, except for the initial cut on $\{c_1, \ldots, c_n, not\ y\}$, applications of *Cut*$[body(\Pi_b^n \cup \Pi_c^n)]$ do not admit immediate contradictions and must thus be cascaded to form a perfect binary tree. Hence, a minimal refutation of $\mathcal{T}_{nomore}$ for $\Pi_b^n \cup \Pi_c^n$ is of size exponential in n.

We have thus shown that the asymptotic sizes of minimal refutations of $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ for $\Pi_b^n \cup \Pi_c^n$ are $O(n)$ and $O(2^n)$, respectively. Hence, $\mathcal{T}_{smodels}$ is not polynomially simulated by $\mathcal{T}_{nomore}$. □

Corollary 3.17. *Tableau calculi $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$ are efficiency-incomparable.*

Proof. This result follows immediately from Proposition 3.15 and 3.16, since they show that neither $\mathcal{T}_{nomore}$ is polynomially simulated by $\mathcal{T}_{smodels}$, nor vice versa. □

Corollary 3.18. *Tableau calculus $\mathcal{T}_{nomore++}$ is exponentially stronger than both $\mathcal{T}_{smodels}$ and $\mathcal{T}_{nomore}$.*

Proof. This result follows immediately from Corollary 3.17, since $\mathcal{T}_{nomore}$ and $\mathcal{T}_{smodels}$ are both polynomially simulated by $\mathcal{T}_{nomore++}$ (any tableau of $\mathcal{T}_{nomore}$ or $\mathcal{T}_{smodels}$ is a tableau of $\mathcal{T}_{nomore++}$ as well), while $\mathcal{T}_{nomore}$ and $\mathcal{T}_{smodels}$ are not polynomially simulated by one another. □

Proposition 3.19. *Tableau calculus $\mathcal{T}_{card}$ is exponentially stronger than $\mathcal{T}_{conj}$.*

Proof. Consider the following family $\{\Pi_c^n \cup \Pi_d^n\}$ of cardinality programs for $n \geq 1$:

$$\Pi_c^n \cup \Pi_d^n \;=\; \{z \leftarrow 1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\} \cup \bigcup\nolimits_{1 \leq i \leq n} \left\{ \begin{array}{l} a_i \leftarrow not\ b_i \\ b_i \leftarrow not\ a_i \end{array} \right\}$$

The domain of assignments $\mathbf{A}$ is $dom(\mathbf{A}) = \{z, \{1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\}\} \cup \bigcup_{1 \leq i \leq n} \{a_i, b_i, 1\{a_i, b_i\}2\}$.[6] We separately investigate minimal refutations of $\mathcal{T}_{card}$ and $\mathcal{T}_{conj}$ for members of $\{\Pi_c^n \cup \Pi_d^n\}$:

($\mathcal{T}_{card}$) An optimal strategy to construct a refutation of $\mathcal{T}_{card}$ for $\Pi_c^n \cup \Pi_d^n$ is as follows:

1. Cut on z, complete the branch for $\boldsymbol{T}z$, using the deterministic tableau rules $N{\downarrow}$ and $FC{\uparrow}$, and deduce $\boldsymbol{F}\{1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\}$ in the branch for $\boldsymbol{F}z$, using the deterministic tableau rule $I{\downarrow}$.
2. Complete the branch containing $\boldsymbol{F}\{1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\}$ (and $\boldsymbol{F}z$), but none of $\boldsymbol{F}1\{a_i, b_i\}2$ for $1 \leq i \leq n$, if it contains $n-1$ entries of the form $\boldsymbol{T}1\{a_i, b_i\}2$, using the deterministic tableau rules $TC{\downarrow}$, $TL_U{\downarrow}$, and $I{\downarrow}$. Otherwise, if there are less than $n-1$ entries of the form $\boldsymbol{T}1\{a_i, b_i\}2$ in the branch, cut on some unassigned $1\{a_i, b_i\}2$ for $1 \leq i \leq n$ and complete the branch for $\boldsymbol{F}1\{a_i, b_i\}2$, using the deterministic tableau rules $TL_U{\downarrow}$ and $I{\downarrow}$.

In a nutshell, a refutation constructed in this way makes use of immediate contradictions obtained when assigning any of the cardinality constraints $1\{a_i, b_i\}2$ to false, so that each application of $\mathit{Cut}[atom(\Pi_c^n \cup \Pi_d^n) \cup conj(\Pi_c^n \cup \Pi_d^n) \cup card(\Pi_c^n \cup \Pi_d^n)]$ yields one branch that is completed without cutting any further. Hence, such a refutation of $\mathcal{T}_{card}$ for $\Pi_c^n \cup \Pi_d^n$ is of size linear in n.

($\mathcal{T}_{conj}$) An optimal strategy to construct a refutation of $\mathcal{T}_{conj}$ for $\Pi_c^n \cup \Pi_d^n$ is as follows:

1. Cut on z, complete the branch for $\boldsymbol{T}z$, using the deterministic tableau rules $N{\downarrow}$ and $FC{\uparrow}$, and deduce $\boldsymbol{F}\{1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\}$ in the branch for $\boldsymbol{F}z$, using the deterministic tableau rule $I{\downarrow}$.
2. Complete any of the branches containing $\boldsymbol{F}\{1\{a_1, b_1\}2, \ldots, 1\{a_n, b_n\}2, not\ z\}$ (and $\boldsymbol{F}z$) if the branch contains $n-1$ entries of the form $\boldsymbol{T}1\{a_i, b_i\}2$, using the deterministic tableau rules $TC{\downarrow}$, $TL_U{\downarrow}$, and $I{\downarrow}$. Otherwise, if there are less than $n-1$ entries of the form $\boldsymbol{T}1\{a_i, b_i\}2$ in a branch, cut on some unassigned a_i for $1 \leq i \leq n$ and deduce $\boldsymbol{T}1\{a_i, b_i\}2$ in the branch for $\boldsymbol{T}a_i$ as well as in the branch for $\boldsymbol{F}a_i$, using the deterministic tableau rules $TLU{\uparrow}$ and $I{\downarrow}$.

[6] For convenience, we take $not\ a_i$ and $not\ b_i$ to be atomic literals, rather than elements of a (singleton) conjunction. The latter would also be possible and, in view of the deterministic tableau rules in Figure 3.5 on Page 37, not affect proof complexity.

As the second step shows, cuts on atoms a_i (or b_i) for $1 \leq i \leq n$ yield symmetric alternatives, since $\boldsymbol{T}1\{a_i, b_i\}2$ is deduced in each of the resulting branches. That is, except for the initial cut on z, applications of $\mathit{Cut}[atom(\Pi_c^n \cup \Pi_d^n) \cup conj(\Pi_c^n \cup \Pi_d^n)]$ do not admit immediate contradictions and must thus be cascaded to form a perfect binary tree. Hence, a minimal refutation of $\mathcal{T}_{conj}$ for $\Pi_c^n \cup \Pi_d^n$ is of size exponential in n.

We have thus shown that the asymptotic sizes of minimal refutations of $\mathcal{T}_{card}$ and $\mathcal{T}_{conj}$ for $\Pi_c^n \cup \Pi_d^n$ are $O(n)$ and $O(2^n)$, respectively. Since $\mathcal{T}_{conj}$ is polynomially simulated by $\mathcal{T}_{card}$, this shows that $\mathcal{T}_{card}$ is exponentially stronger than $\mathcal{T}_{conj}$. □

Finally, we case by case show that the application of a tableau rule $R{\downarrow}$ can be simulated by means of *Cut* and $R{\uparrow}$, so that the inclusion or exclusion of $R{\downarrow}$ cannot (alone) be responsible for an exponential separation between tableau calculi.

Proposition 3.20. *Let Π be a disjunctive program, $\mathcal{T}$ a tableau calculus containing any subset of the tableau rules (a)–(v), and $\mathcal{T}'$ an approximation of $\mathcal{T}$.*

If $\mathit{Cut}[\Gamma] \in \mathcal{T}'$ such that $atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \subseteq \Gamma$, then we have that $\mathcal{T}$ is polynomially simulated by $\mathcal{T}'$.

Proof. Assume that $\mathit{Cut}[\Gamma] \in \mathcal{T}'$ such that $atom(\Pi) \cup conj(\Pi) \cup card(\Pi) \subseteq \Gamma$. Then, we show that deducing an entry σ by a tableau rule $R{\downarrow}$ can be simulated by cutting on $var(\sigma)$ and completing the branch for $\overline{\sigma}$ by an application of $R{\uparrow}$. To demonstrate this, we consider all tableau rules $R{\downarrow}$ and show that $\mathbf{A} \cup D_{\{R\uparrow\}}(\Pi, \mathbf{A} \cup \{\overline{\sigma}\})$ is contradictory if $\sigma \in D_{\{R\downarrow\}}(\Pi, \mathbf{A})$:

($I{\downarrow}$) If $\boldsymbol{f}\beta \in D_{\{I\downarrow\}}(\Pi, \mathbf{A})$, we have that $\boldsymbol{f}\alpha \in \mathbf{A}$. Since $\boldsymbol{t}\alpha \in D_{\{I\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{t}\beta\})$, it holds that $\mathbf{A} \cup D_{\{I\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{t}\beta\})$ is contradictory.

($N{\downarrow}$) If $\sigma \in D_{\{N\downarrow\}}(\Pi, \mathbf{A})$, we have that $\sigma \in \{\boldsymbol{t}\beta\} \cup min_{\mathbf{A}}(\alpha, \{p\}) \cup max_{\mathbf{A}}(\beta, \emptyset)$ for some $p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ such that $sup_{\mathbf{A}}(\Pi, \{p\}, \emptyset) = \{\alpha \leftarrow \beta\}$. For $\sigma = \boldsymbol{t}\beta$, we get that $(\alpha \leftarrow \beta) \notin sup_{\mathbf{A} \cup \{\boldsymbol{f}\beta\}}(\Pi, \{p\}, \emptyset) = \{(\alpha' \leftarrow \beta') \in \Pi \mid \boldsymbol{f}\beta' \notin \mathbf{A} \cup \{\boldsymbol{f}\beta\}, \overleftarrow{sup}_{\mathbf{A} \cup \{\boldsymbol{f}\beta\}}(\alpha', \{p\}), \overrightarrow{sup}_{\mathbf{A} \cup \{\boldsymbol{f}\beta\}}(\beta', \emptyset)\}$. For $\sigma \in min_{\mathbf{A}}(\alpha, \{p\})$ or $\sigma \in max_{\mathbf{A}}(\beta, \emptyset)$, Lemma B.10 yields that $\overleftarrow{sup}_{\mathbf{A} \cup \{\overline{\sigma}\}}(\alpha, \{p\})$ or $\overrightarrow{sup}_{\mathbf{A} \cup \{\overline{\sigma}\}}(\beta, \emptyset)$, respectively, does not hold, which as with $\sigma = \boldsymbol{t}\beta$ implies that $(\alpha \leftarrow \beta) \notin sup_{\mathbf{A} \cup \{\overline{\sigma}\}}(\Pi, \{p\}, \emptyset)$. By Lemma B.11, we further conclude that $sup_{\mathbf{A} \cup \{\overline{\sigma}\}}(\Pi, \{p\}, \emptyset) \subseteq sup_{\mathbf{A}}(\Pi, \{p\}, \emptyset) \setminus \{\alpha \leftarrow \beta\} = \emptyset$. That is, $\boldsymbol{F}p \in D_{\{N\uparrow\}}(\Pi, \mathbf{A} \cup \{\overline{\sigma}\})$ for some $p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$, so that $\mathbf{A} \cup D_{\{N\uparrow\}}(\Pi, \mathbf{A} \cup \{\overline{\sigma}\})$ is contradictory.

($U{\downarrow}$) If $\sigma \in D_{\{U\downarrow\}}(\Pi, \mathbf{A})$, we have that $\sigma \in \{\boldsymbol{t}\beta\} \cup min_{\mathbf{A}}(\alpha, S) \cup max_{\mathbf{A}}(\beta, S)$ for some $S \subseteq atom(\Pi)$ such that $\mathbf{A}^{\boldsymbol{T}} \cap S \neq \emptyset$ and $sup_{\mathbf{A}}(\Pi, S, S) = \{\alpha \leftarrow \beta\}$. For $\sigma = \boldsymbol{t}\beta$, we get that $(\alpha \leftarrow \beta) \notin sup_{\mathbf{A} \cup \{\boldsymbol{f}\beta\}}(\Pi, S, S) = \{(\alpha' \leftarrow \beta') \in \Pi \mid \boldsymbol{f}\beta' \notin \mathbf{A} \cup \{\boldsymbol{f}\beta\}, \overleftarrow{sup}_{\mathbf{A} \cup \{\boldsymbol{f}\beta\}}(\alpha', S), \overrightarrow{sup}_{\mathbf{A} \cup \{\boldsymbol{f}\beta\}}(\beta', S)\}$. For $\sigma \in min_{\mathbf{A}}(\alpha, S)$ or $\sigma \in max_{\mathbf{A}}(\beta, S)$, Lemma B.10 yields that $\overleftarrow{sup}_{\mathbf{A} \cup \{\overline{\sigma}\}}(\alpha, S)$ or $\overrightarrow{sup}_{\mathbf{A} \cup \{\overline{\sigma}\}}(\beta, S)$, respectively, does not hold, which as with $\sigma = \boldsymbol{t}\beta$ implies that $(\alpha \leftarrow \beta) \notin sup_{\mathbf{A} \cup \{\overline{\sigma}\}}(\Pi, S, S)$. By Lemma B.11, we further conclude that $sup_{\mathbf{A} \cup \{\overline{\sigma}\}}(\Pi, S, S) \subseteq sup_{\mathbf{A}}(\Pi, S, S) \setminus \{\alpha \leftarrow \beta\} = \emptyset$. That is, $\boldsymbol{F}p \in D_{\{U\uparrow\}}(\Pi, \mathbf{A} \cup \{\overline{\sigma}\})$ for some $p \in \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$, so that $\mathbf{A} \cup D_{\{U\uparrow\}}(\Pi, \mathbf{A} \cup \{\overline{\sigma}\})$ is contradictory.

($TC\downarrow$**)** If $\boldsymbol{f}l_i \in D_{\{TC\downarrow\}}(\Pi, \mathbf{A})$, we have that $\{\boldsymbol{F}C, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_{i-1}, \boldsymbol{t}l_{i+1}, \ldots, \boldsymbol{t}l_n\} \subseteq \mathbf{A}$ for $C = \{l_1, \ldots, l_{i-1}, l_i, l_{i+1}, \ldots, l_n\} \in \mathit{conj}(\Pi)$. Since $\boldsymbol{T}C \in D_{\{TC\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{t}l_i\})$, it holds that $\mathbf{A} \cup D_{\{TC\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{t}l_i\})$ is contradictory.

($FC\downarrow$**)** If $\boldsymbol{t}l_i \in D_{\{FC\downarrow\}}(\Pi, \mathbf{A})$, we have that $\boldsymbol{T}C \in \mathbf{A}$ for $C = \{l_1, \ldots, l_i, \ldots, l_n\} \in \mathit{conj}(\Pi)$. Since $\boldsymbol{F}C \in D_{\{FC\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{f}l_i\})$, it holds that $\mathbf{A} \cup D_{\{FC\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{f}l_i\})$ is contradictory.

($TLU\downarrow$**)** If $\boldsymbol{f}l_j \in D_{\{TLU\downarrow\}}(\Pi, \mathbf{A})$, we have that $\{\boldsymbol{F}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_{j-1}, \boldsymbol{f}l_{k+1}, \ldots, \boldsymbol{f}l_n\} \subseteq \mathbf{A}$ for $B = j\{l_1, \ldots, l_j, \ldots, l_{k+1}, \ldots, l_n\}k \in \mathit{card}(\Pi)$. Since $\boldsymbol{T}B \in D_{\{TLU\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{t}l_j\})$, it holds that $\mathbf{A} \cup D_{\{TLU\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{t}l_j\})$ is contradictory.

($T_L U\downarrow$**)** If $\boldsymbol{t}l_{k+1} \in D_{\{T_L U\downarrow\}}(\Pi, \mathbf{A})$, we have that $\{\boldsymbol{F}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_j, \boldsymbol{f}l_{k+2}, \ldots, \boldsymbol{f}l_n\} \subseteq \mathbf{A}$ for $B = j\{l_1, \ldots, l_j, \ldots, l_{k+1}, \ldots, l_n\}k \in \mathit{card}(\Pi)$. Since $\boldsymbol{T}B \in D_{\{TLU\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{f}l_{k+1}\})$, it holds that $\mathbf{A} \cup D_{\{TLU\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{f}l_{k+1}\})$ is contradictory.

($FL\downarrow$**)** If $\boldsymbol{t}l_j \in D_{\{FL\downarrow\}}(\Pi, \mathbf{A})$, we have that $\{\boldsymbol{T}B, \boldsymbol{f}l_{j+1}, \ldots, \boldsymbol{f}l_n\} \subseteq \mathbf{A}$ for $B = j\{l_1, \ldots, l_j, \ldots, l_n\}k \in \mathit{card}(\Pi)$. Since $\boldsymbol{F}B \in D_{\{FL\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{f}l_j\})$, it holds that $\mathbf{A} \cup D_{\{FL\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{f}l_j\})$ is contradictory.

($FU\downarrow$**)** If $\boldsymbol{f}l_{k+1} \in D_{\{FU\downarrow\}}(\Pi, \mathbf{A})$, we have that $\{\boldsymbol{T}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_k\} \subseteq \mathbf{A}$ for $B = j\{l_1, \ldots, l_{k+1}, \ldots, l_n\}k \in \mathit{card}(\Pi)$. Since $\boldsymbol{F}B \in D_{\{FU\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{t}l_{k+1}\})$, it holds that $\mathbf{A} \cup D_{\{FU\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{t}l_{k+1}\})$ is contradictory.

($TD\downarrow$**)** If $\boldsymbol{f}l_i \in D_{\{TD\downarrow\}}(\Pi, \mathbf{A})$, we have that $\boldsymbol{F}D \in \mathbf{A}$ for $D = \{l_1; \ldots; l_i; \ldots; l_n\} \in \mathit{disj}(\Pi)$. Since $\boldsymbol{T}D \in D_{\{TD\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{t}l_i\})$, it holds that $\mathbf{A} \cup D_{\{TD\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{t}l_i\})$ is contradictory.

($FD\downarrow$**)** If $\boldsymbol{t}l_i \in D_{\{FD\downarrow\}}(\Pi, \mathbf{A})$, we have that $\{\boldsymbol{T}D, \boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_{i-1}, \boldsymbol{f}l_{i+1}, \ldots, \boldsymbol{f}l_n\} \subseteq \mathbf{A}$ for $D = \{l_1; \ldots; l_{i-1}; l_i; l_{i+1}; \ldots; l_n\} \in \mathit{disj}(\Pi)$. Since $\boldsymbol{F}D \in D_{\{FD\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{f}l_i\})$, it holds that $\mathbf{A} \cup D_{\{FD\uparrow\}}(\Pi, \mathbf{A} \cup \{\boldsymbol{f}l_i\})$ is contradictory.

We have thus shown that deducing σ by a tableau rule $R\downarrow$ can be simulated by means of applying *Cut* and $R\uparrow$. As each such simulation introduces only two additional entries, $\overline{\sigma}$ and the complement of some entry belonging to the branch at hand, every tableau of $\mathcal{T}$ can be transformed into a tableau of $\mathcal{T}'$ having approximately similar size, provided that the *Cut* applications needed for simulations are admitted by $\mathcal{T}'$. In fact, the variable of an entry deducible by a tableau rule $R\downarrow$ cannot be a disjunction, so that all simulations are possible if $\mathit{Cut}[\Gamma] \in \mathcal{T}'$ such that $\mathit{atom}(\Pi) \cup \mathit{conj}(\Pi) \cup \mathit{card}(\Pi) \subseteq \Gamma$. □

We have thus proven all formal results presented in Chapter 3.

B.3 Chapter 4

We present proofs of results by sections, dealing with a characterization of answer sets in terms of nogoods, a decision algorithm for answer set existence, and enumeration algorithms for answer sets and their projections to a subvocabulary, respectively.

B.3.1 Section 4.1

To begin with, we show Proposition 4.1 on the correspondence between models of $Comp(\Pi)$ and solutions for Δ_Π.

Proposition 4.1. *Let Π be a normal program and $X \subseteq atom(\Pi) \cup body(\Pi)$.*

Then, we have that $(X \cap atom(\Pi)) \cup \{p_B \mid B \in X \cap body(\Pi)\}$ is a model of $Comp(\Pi)$ iff $\{\boldsymbol{T}v \mid v \in X\} \cup \{\boldsymbol{F}v \mid v \in (atom(\Pi) \cup body(\Pi)) \setminus X\}$ is a solution for Δ_Π.

Proof. We separately show the implications of the statement:

(⇒) Assume that $M = (X \cap atom(\Pi)) \cup \{p_B \mid B \in X \cap body(\Pi)\}$ is a model of $Comp(\Pi)$, and let $\mathbf{A} = \{\boldsymbol{T}v \mid v \in X\} \cup \{\boldsymbol{F}v \mid v \in (atom(\Pi) \cup body(\Pi)) \setminus X\}$.

Then, for every $p \in atom(\Pi)$ and $body_\Pi(p) = \{B_1, \ldots, B_k\}$,

$$M \models p \leftrightarrow (p_{B_1} \vee \cdots \vee p_{B_k})$$

yields that $p \in X$ iff $\{B_1, \ldots, B_k\} \cap X \neq \emptyset$, which in turn implies that $\{\boldsymbol{T}p, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \not\subseteq \mathbf{A}, \{\boldsymbol{F}p, \boldsymbol{T}B_1\} \not\subseteq \mathbf{A}, \ldots, \{\boldsymbol{F}p, \boldsymbol{T}B_k\} \not\subseteq \mathbf{A}$.

Furthermore, for every $B = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\} \in body(\Pi)$,

$$M \models p_B \leftrightarrow (p_1 \wedge \cdots \wedge p_m \wedge \neg p_{m+1} \wedge \cdots \wedge \neg p_n)$$

yields that $B \in X$ iff $\{p_1, \ldots, p_m, p_{m+1}, \ldots, p_n\} \cap X = \{p_1, \ldots, p_m\}$, which in turn implies that $\{\boldsymbol{F}B, \boldsymbol{T}p_1, \ldots, \boldsymbol{T}p_m, \boldsymbol{F}p_{m+1}, \ldots, \boldsymbol{F}p_n\} \not\subseteq \mathbf{A}, \{\boldsymbol{T}B, \boldsymbol{F}p_1\} \not\subseteq \mathbf{A}, \ldots, \{\boldsymbol{T}B, \boldsymbol{F}p_m\} \not\subseteq \mathbf{A}, \{\boldsymbol{T}B, \boldsymbol{T}p_{m+1}\} \not\subseteq \mathbf{A}, \ldots, \{\boldsymbol{T}B, \boldsymbol{T}p_n\} \not\subseteq \mathbf{A}$.

We have thus shown that none of the nogoods in Δ_Π is contained in $\mathbf{A}$, so that $\mathbf{A}$ is a solution for Δ_Π.

(⇐) Assume that $\mathbf{A} = \{\boldsymbol{T}v \mid v \in X\} \cup \{\boldsymbol{F}v \mid v \in (atom(\Pi) \cup body(\Pi)) \setminus X\}$ is a solution for Δ_Π, and let $M = (X \cap atom(\Pi)) \cup \{p_B \mid B \in X \cap body(\Pi)\}$.

Then, for every $p \in atom(\Pi)$ and $body_\Pi(p) = \{B_1, \ldots, B_k\}$, the fact that $\{\boldsymbol{T}p, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \not\subseteq \mathbf{A}$ yields that $p \notin X$ if $\{B_1, \ldots, B_k\} \cap X = \emptyset$. Moreover, since $\{\boldsymbol{F}p, \boldsymbol{T}B_1\} \not\subseteq \mathbf{A}, \ldots, \{\boldsymbol{F}p, \boldsymbol{T}B_k\} \not\subseteq \mathbf{A}$, we have that $p \in X$ if $\{B_1, \ldots, B_k\} \cap X \neq \emptyset$. From this, we conclude that $p \in X$ iff $\{B_1, \ldots, B_k\} \cap X \neq \emptyset$, so that

$$M \models p \leftrightarrow (p_{B_1} \vee \cdots \vee p_{B_k}).$$

Furthermore, for every $B = \{p_1, \ldots, p_m, not\ p_{m+1}, \ldots, not\ p_n\} \in body(\Pi)$, the fact that $\{\boldsymbol{F}B, \boldsymbol{T}p_1, \ldots, \boldsymbol{T}p_m, \boldsymbol{F}p_{m+1}, \ldots, \boldsymbol{F}p_n\} \not\subseteq \mathbf{A}$ yields that $B \in X$ if $\{p_1, \ldots, p_m, p_{m+1}, \ldots, p_n\} \cap X = \{p_1, \ldots, p_m\}$. Moreover, since $\{\boldsymbol{T}B, \boldsymbol{F}p_1\} \not\subseteq \mathbf{A}, \ldots, \{\boldsymbol{T}B, \boldsymbol{F}p_m\} \not\subseteq \mathbf{A}, \{\boldsymbol{T}B, \boldsymbol{T}p_{m+1}\} \not\subseteq \mathbf{A}, \ldots, \{\boldsymbol{T}B, \boldsymbol{T}p_n\} \not\subseteq \mathbf{A}$, we have that $B \notin X$ if $\{p_1, \ldots, p_m, p_{m+1}, \ldots, p_n\} \cap X \neq \{p_1, \ldots, p_m\}$. From this, we conclude that $B \in X$ iff $\{p_1, \ldots, p_m, p_{m+1}, \ldots, p_n\} \cap X = \{p_1, \ldots, p_m\}$, so that

$$M \models p_B \leftrightarrow (p_1 \wedge \cdots \wedge p_m \wedge \neg p_{m+1} \wedge \cdots \wedge \neg p_n).$$

We have thus shown that M satisfies all formulas of $Comp(\Pi)$, so that M is a model of $Comp(\Pi)$. □

For proving Theorem 4.3 on the one-to-one correspondence between answer sets of a tight program Π and solutions for Δ_Π, we make use of Lemma 4.2, establishing that any solution for Δ_Π is uniquely determined by its entries over atoms.

Lemma 4.2. *Let* Π *be a normal program and* $X \subseteq atom(\Pi)$.
Then, we have that

$$\begin{array}{rcl} \mathbf{A} & = & \{\boldsymbol{T}p \mid p \in X\} \cup \{\boldsymbol{F}p \mid p \in atom(\Pi) \setminus X\} \\ & \cup & \{\boldsymbol{T}B \mid B \in body(\Pi), B^+ \subseteq X, B^- \cap X = \emptyset\} \\ & \cup & \{\boldsymbol{F}B \mid B \in body(\Pi), (B^+ \cap (atom(\Pi) \setminus X)) \cup (B^- \cap X) \neq \emptyset\} \end{array}$$

is the unique solution for

$$\bigcup_{B \in body(\Pi), B=\{l_1,\ldots,l_n\}} \{\{\boldsymbol{F}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\}, \{\boldsymbol{T}B, \boldsymbol{f}l_1\}, \ldots, \{\boldsymbol{T}B, \boldsymbol{f}l_n\}\} \quad \subseteq \quad \Delta_\Pi$$

such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$.

Proof. For every $B = \{l_1, \ldots, l_n\} \in body(\Pi)$, one of the following cases applies:

($B^+ \subseteq X$ and $B^- \cap X = \emptyset$) We have that $\{\boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\} = \{\boldsymbol{T}p \mid p \in B^+\} \cup \{\boldsymbol{F}p \mid p \in B^-\} \subseteq \mathbf{A}$. Since $\{\boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_n\} \cap \mathbf{A} = \emptyset$, it holds that $\{\boldsymbol{T}B, \boldsymbol{f}l_1\} \not\subseteq \mathbf{A}$, $\ldots, \{\boldsymbol{T}B, \boldsymbol{f}l_n\} \not\subseteq \mathbf{A}$. Furthermore, $\boldsymbol{T}B \in \mathbf{A}$ and $\boldsymbol{F}B \notin \mathbf{A}$ make sure that $\{\boldsymbol{F}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\} \not\subseteq \mathbf{A}$, where $\{\boldsymbol{F}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\} \setminus \mathbf{A} = \{\boldsymbol{F}B\}$.

($(B^+ \cap (atom(\Pi) \setminus X)) \cup (B^- \cap X) \neq \emptyset$) We have that $\{\boldsymbol{f}l_1, \ldots, \boldsymbol{f}l_n\} \cap \mathbf{A} = (\{\boldsymbol{F}p \mid p \in B^+\} \cup \{\boldsymbol{T}p \mid p \in B^-\}) \cap \mathbf{A} \neq \emptyset$. Since $\{\boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\} \not\subseteq \mathbf{A}$, it holds that $\{\boldsymbol{F}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\} \not\subseteq \mathbf{A}$. Furthermore, $\boldsymbol{F}B \in \mathbf{A}$ and $\boldsymbol{T}B \notin \mathbf{A}$ make sure that $\{\boldsymbol{T}B, \boldsymbol{f}l_1\} \not\subseteq \mathbf{A}, \ldots, \{\boldsymbol{T}B, \boldsymbol{f}l_n\} \not\subseteq \mathbf{A}$, where $\{\boldsymbol{T}B, \boldsymbol{f}l_i\} \setminus \mathbf{A} = \{\boldsymbol{T}B\}$ for some $1 \leq i \leq n$.

The above cases show that, for every $B = \{l_1, \ldots, l_n\} \in body(\Pi)$, none of the nogoods $\{\boldsymbol{F}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\}, \{\boldsymbol{T}B, \boldsymbol{f}l_1\}, \ldots, \{\boldsymbol{T}B, \boldsymbol{f}l_n\}$ is contained in $\mathbf{A}$, so that $\mathbf{A}$ is a solution for $\bigcup_{B \in body(\Pi), B=\{l_1,\ldots,l_n\}} \{\{\boldsymbol{F}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\}, \{\boldsymbol{T}B, \boldsymbol{f}l_1\}, \ldots, \{\boldsymbol{T}B, \boldsymbol{f}l_n\}\} \subseteq \Delta_\Pi$. On the other hand, for each $B = \{l_1, \ldots, l_n\} \in body(\Pi)$, there is some $\delta \in \{\{\boldsymbol{F}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\}, \{\boldsymbol{T}B, \boldsymbol{f}l_1\}, \ldots, \{\boldsymbol{T}B, \boldsymbol{f}l_n\}\}$ such that $\delta \setminus \mathbf{A} = \{\boldsymbol{F}B\}$ or $\delta \setminus \mathbf{A} = \{\boldsymbol{T}B\}$, respectively. Hence, there is no solution $\mathbf{B} \neq \mathbf{A}$ for $\bigcup_{B \in body(\Pi), B=\{l_1,\ldots,l_n\}} \{\{\boldsymbol{F}B, \boldsymbol{t}l_1, \ldots, \boldsymbol{t}l_n\}, \{\boldsymbol{T}B, \boldsymbol{f}l_1\}, \ldots, \{\boldsymbol{T}B, \boldsymbol{f}l_n\}\} \subseteq \Delta_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi) = X$. □

Theorem 4.3. *Let* Π *be a tight program and* $X \subseteq atom(\Pi)$.
Then, we have that X *is an answer set of* Π *iff*

$$\begin{array}{rcl} \mathbf{A} & = & \{\boldsymbol{T}p \mid p \in X\} \cup \{\boldsymbol{F}p \mid p \in atom(\Pi) \setminus X\} \\ & \cup & \{\boldsymbol{T}B \mid B \in body(\Pi), B^+ \subseteq X, B^- \cap X = \emptyset\} \\ & \cup & \{\boldsymbol{F}B \mid B \in body(\Pi), (B^+ \cap (atom(\Pi) \setminus X)) \cup (B^- \cap X) \neq \emptyset\} \end{array}$$

is the unique solution for Δ_Π *such that* $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$.

Proof. By Lemma 4.2, there is a subset of Δ_Π for which $\mathbf{A}$ is the unique solution such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$. Along with Proposition 4.1, we conclude that $M \cap atom(\Pi) = X$ for some model M of $Comp(\Pi)$ iff $\mathbf{A}$ is the unique solution for Δ_Π such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$. Finally, by Theorem 3.2 in [68], showing that models of $Comp(\Pi)$ coincide with answer sets of Π if Π is tight,[7] we conclude that X is an answer set of Π iff $\mathbf{A}$ is the unique solution for Δ_Π such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$. □

[7]Note that $Comp(\Pi)$ defines the propositions standing for bodies in terms of atoms. Hence, $Comp(\Pi)$ is a conservative extension of the completion of Π, originally described without propositions for bodies [37].

In order to extend the above correspondence result to non-tight programs Π and solutions for $\Delta_\Pi \cup \Lambda_\Pi$, we first provide Lemma B.16, linking unfounded sets to the nogoods in Λ_Π. The proof of Theorem 4.4 makes use of this connection by exploiting the characterization of answer set in terms of unfounded sets given in Corollary 2.4.

Lemma B.16. *Let Π be a normal program and $\mathbf{A}$ an assignment.*

Then, we have that $\delta \subseteq \mathbf{A}$ for some $\delta \in \Lambda_\Pi$ iff there is an unfounded set U of Π w.r.t. $\mathbf{A}$ such that $U \cap \mathbf{A}^{\boldsymbol{T}} \neq \emptyset$.

Proof. Every $\delta \in \Lambda_\Pi$ is of the form $\{\boldsymbol{T}p, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\}$, where $p \in U$ for some $U \subseteq atom(\Pi)$ such that $EB_\Pi(U) = \{B_1, \ldots, B_k\}$. Hence, we have that $\delta \subseteq \mathbf{A}$ for some $\delta \in \Lambda_\Pi$ iff there is some $U \subseteq atom(\Pi)$ such that $\{\boldsymbol{T}p, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \subseteq \mathbf{A}$ for $p \in U$ and $EB_\Pi(U) = \{B_1, \ldots, B_k\}$ iff there is an unfounded set U of Π w.r.t. $\mathbf{A}$ such that $U \cap \mathbf{A}^{\boldsymbol{T}} \neq \emptyset$. □

Theorem 4.4. *Let Π be a normal program and $X \subseteq atom(\Pi)$.*

Then, we have that X is an answer set of Π iff

$$\begin{aligned}\mathbf{A} \;=\;& \{\boldsymbol{T}p \mid p \in X\} \cup \{\boldsymbol{F}p \mid p \in atom(\Pi) \setminus X\}\\ \cup\;& \{\boldsymbol{T}B \mid B \in body(\Pi), B^+ \subseteq X, B^- \cap X = \emptyset\}\\ \cup\;& \{\boldsymbol{F}B \mid B \in body(\Pi), (B^+ \cap (atom(\Pi) \setminus X)) \cup (B^- \cap X) \neq \emptyset\}\end{aligned}$$

is the unique solution for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$.

Proof. By Lemma 4.2, there is a subset of Δ_Π for which $\mathbf{A}$ is the unique solution such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$. Along with Proposition 4.1, we conclude that $M \cap atom(\Pi) = X$ for some model M of $Comp(\Pi)$ iff $\mathbf{A}$ is the unique solution for Δ_Π such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$. For any model M of $Comp(\Pi)$, it is clear that $M \cap atom(\Pi)$ is a model of Π, that is, $head(r) \in M$, $body(r)^+ \not\subseteq M$, or $body(r)^- \cap M \neq \emptyset$ holds for every rule $r \in \Pi$. Hence, if $\mathbf{A}$ is the unique solution for Δ_Π such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$, then $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is a model of Π. In addition, we have that $\mathbf{A}$ is body-synchronized for Π according to Definition 2.3 because $\mathbf{A}^{\boldsymbol{F}} \cap body(\Pi) = \{B \in body(\Pi) \mid (B^+ \cap (atom(\Pi) \setminus X)) \cup (B^- \cap X) \neq \emptyset\} = \{B \in body(\Pi) \mid (B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\}$. We exploit these properties for showing the implications of the statement:

(⇒) Assume that X is an answer set of Π. Then, by Corollary 1 in [174], we have that $M \cap atom(\Pi) = X$ for some model M of $Comp(\Pi)$. (See Footnote 7 on Page 154 for remarks on the role of propositions standing for bodies in $Comp(\Pi)$.) That is, $\mathbf{A}$ is the unique solution for Δ_Π such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$. Since $\mathbf{A}$ is body-synchronized for Π and $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is a model of Π, by Corollary 2.4, we conclude that $U \cap \mathbf{A}^{\boldsymbol{T}} = \emptyset$ holds for every unfounded set U of Π w.r.t. $\mathbf{A}$. Hence, by Lemma B.16, $\mathbf{A}$ is a solution for Λ_Π and the unique solution for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$.

(⇐) Assume that $\mathbf{A}$ is the unique solution for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi) = X$. Then, by Lemma B.16, we have that $U \cap \mathbf{A}^{\boldsymbol{T}} = \emptyset$ holds for every unfounded set U of Π w.r.t. $\mathbf{A}$. Since $\mathbf{A}$ is body-synchronized for Π and $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is a model of Π, by Corollary 2.4, we conclude that X is an answer set of Π. □

We have thus proven the formal results presented in Section 4.1.

B.3.2 Section 4.3

We start by showing fundamental properties of UNFOUNDEDSET in Algorithm 4.3 on Page 68, where Lemma 4.5 and 4.6 establish invariants that are crucial for its soundness and completeness, stated in Theorem 4.7.

Lemma 4.5. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment that is body-saturated for* Π.

If UNFOUNDEDSET($\Pi, \mathbf{A}$) *is invoked on a valid source pointer configuration, then we have that the source pointer configuration remains valid throughout the execution of* UNFOUNDEDSET($\Pi, \mathbf{A}$).

Proof. Assume that UNFOUNDEDSET($\Pi, \mathbf{A}$) is invoked on a valid source pointer configuration. Then, an invalid source pointer configuration could in principle be obtained only in Line 13 of Algorithm 4.3, where $source(q)$ is set for some (cyclic) $q \in atom(\Pi)$. However, by induction on executions of Line 13, we show that the source pointer configuration remains valid:

(Base case) Since the given source pointer configuration is valid and $\mathbf{A}$ is body-saturated for Π,[8] after finishing the loop in Line 2–5 of Algorithm 4.3, we have that $source(p) \in body_\Pi(p) \setminus \mathbf{A}^{\boldsymbol{F}}$ and $source(p)^+ \cap (\mathbf{A}^{\boldsymbol{F}} \cup (scc(p) \cap S)) = \emptyset$ hold for every cyclic $p \in atom(\Pi) \setminus (\mathbf{A}^{\boldsymbol{F}} \cup S)$. For the atoms C of any non-trivial strongly connected component of $DG(\Pi)$, this implies that $\bigcup_{p \in C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup S)} (source(p)^+ \cap C) \subseteq C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup S)$. In words, the source pointers of atoms in C that are neither false in $\mathbf{A}$ nor in the scope S do not contain any atom of C that is false in $\mathbf{A}$ or in the scope S.

(Induction step) Let $q \in U$ be any cyclic atom such that the condition in Line 12 of Algorithm 4.3 applies to q, and let $C = scc(q)$. Then, in view of the choice of some $p \in S$ in Line 6 along with Line 7 and 14–16, manipulating the contents of U and S, respectively, we have that $U \subseteq C \cap S$, which yields that $q \in C \cap S$. Furthermore, assume that the source pointer configuration is valid and that $\bigcup_{p \in C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup S)} (source(p)^+ \cap C) \subseteq C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup S)$ holds before setting $source(q)$ to some $B \in body_\Pi(q)$ in Line 13. In terms of the subgraph of $DG(\Pi)$ containing every cyclic $p \in atom(\Pi)$ along with edges (p, p') for all $p' \in source(p)^+ \cap scc(p)$, $\bigcup_{p \in C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup S)} (source(p)^+ \cap C) \subseteq C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup S)$ means that it does not contain any edge from an atom in $C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup S)$ to an atom in $C \cap (\mathbf{A}^{\boldsymbol{F}} \cup S)$. For B, since $\mathbf{A}$ is body-saturated for Π, the condition $B \in EB_\Pi(U) \setminus \mathbf{A}^{\boldsymbol{F}}$ in Line 10 makes sure that $B^+ \cap \mathbf{A}^{\boldsymbol{F}} = \emptyset$, and $B^+ \cap (C \cap S) = \emptyset$ is verified in Line 11. Hence, we have that $B^+ \cap C \subseteq C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup S)$, so that, for all edges (q, p) from q to atoms $p \in B^+ \cap C$, it holds that $p \in C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup S)$. As we have seen above that $q \in C \cap S$ is not reached from atoms in $C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup S)$, we conclude that the subgraph of $DG(\Pi)$ containing every cyclic $p \in atom(\Pi)$ along with edges (p, p') for all $p' \in source(p)^+ \cap scc(p)$ remains acyclic after setting $source(q)$ to B in Line 13. This shows that the source pointer configuration obtained by executing Line 13 is in turn valid. Finally, we have that the induction hypothesis still holds for $S \setminus \{q\}$ constructed in Line 15, that is, $\bigcup_{p \in C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup (S \setminus \{q\}))} (source(p)^+ \cap C) = \big(\bigcup_{p \in C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup S)} (source(p)^+ \cap C)\big) \cup (B^+ \cap C) \subseteq C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup S) \subseteq C \setminus (\mathbf{A}^{\boldsymbol{F}} \cup (S \setminus \{q\}))$.

[8] Recall that $\mathbf{A}$ is body-saturated for Π if $\{B \in body(\Pi) \mid (B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset\} \subseteq \mathbf{A}^{\boldsymbol{F}}$.

We have thus shown that a valid source pointer configuration cannot be invalidated when invoking UNFOUNDEDSET($\Pi, \mathbf{A}$) with an assignment $\mathbf{A}$ that is body-saturated for Π. □

Lemma 4.6. *Let Π be a normal program and $\mathbf{A}$ an assignment that is atom-saturated for Π.*

If UNFOUNDEDSET($\Pi, \mathbf{A}$) *is invoked on a valid source pointer configuration, then we have that every unfounded set $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ of Π w.r.t. $\mathbf{A}$ such that all $p \in U$ belong to the same strongly connected component of $DG(\Pi)$ is contained in S whenever Line 6 of Algorithm 4.3 is entered.*

Proof. Assume that UNFOUNDEDSET($\Pi, \mathbf{A}$) is invoked on a valid source pointer configuration. Then, let $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ be any unfounded set of Π w.r.t. $\mathbf{A}$ such that all $p \in U$ belong to the same strongly connected component of $DG(\Pi)$. Since $U \cap \mathbf{A}^{\boldsymbol{F}} = \emptyset$ and $\mathbf{A}$ is atom-saturated for Π,[9] we have that $body_\Pi(p) \not\subseteq \mathbf{A}^{\boldsymbol{F}}$ for every $p \in U$, while $EB_\Pi(U) \subseteq \mathbf{A}^{\boldsymbol{F}}$ implies that $B^+ \cap U \neq \emptyset$ for each $B \in body_\Pi(p) \setminus \mathbf{A}^{\boldsymbol{F}}$. That is, all $p \in U$ are cyclic, and $source(p) \in body_\Pi(p) \cup \{\bot\}$ holds because the given source pointer configuration is valid. By induction on executions of the test in Line 6 of Algorithm 4.3, we show that $U \not\subseteq S$ is impossible whenever Line 6 is entered:

(Base case) For the sake of contradiction, assume that $U \not\subseteq S$ after finishing the loop in Line 2–5 of Algorithm 4.3. Then, due to Line 1 of Algorithm 4.3, for each $p \in U \setminus S$, we have that $source(p) \notin \mathbf{A}^{\boldsymbol{F}} \cup \{\bot\}$, which further implies that $source(p) \in body_\Pi(p) \setminus \mathbf{A}^{\boldsymbol{F}}$ and $source(p)^+ \cap U \neq \emptyset$. Moreover, the condition $source(p)^+ \cap (scc(p) \cap S) \neq \emptyset$ in Line 3 does not apply to $source(p)$, which yields that $source(p)^+ \cap (U \cap S) = \emptyset$ and $source(p)^+ \cap (U \setminus S) \neq \emptyset$. Since $U \setminus S$ is finite and each atom of $U \setminus S$ has some successor belonging to $U \setminus S$ in the subgraph of $DG(\Pi)$ containing every cyclic $p \in atom(\Pi)$ along with edges (p, q) for all $q \in source(p)^+ \cap scc(p)$, we conclude that this subgraph cannot be acyclic, which is a contradiction to the assumption that UNFOUNDEDSET($\Pi, \mathbf{A}$) is invoked on a valid source pointer configuration.

(Induction step) For the sake of contradiction, assume that $U \subseteq S$ at the beginning of an iteration of the loop in Line 6–17 of Algorithm 4.3, but $U \not\subseteq S$ when Line 6 is reentered after finishing the iteration. In this iteration, the elements of $U \setminus S$ must have (successively) been removed from S in Line 15. In particular, some $q \in U \setminus S$ has been removed from S before any other atom of U. To achieve this, the condition in Line 11 must have applied to some $B \in body_\Pi(q) \setminus \mathbf{A}^{\boldsymbol{F}}$, which yields that $B^+ \cap (scc(q) \cap U') = \emptyset$ for some superset U' of U. Since $U \subseteq scc(q)$, this implies that $B^+ \cap U = \emptyset$, which is a contradiction to the assumption that U is an unfounded set of Π w.r.t. $\mathbf{A}$.

We have thus shown that, if UNFOUNDEDSET($\Pi, \mathbf{A}$) is invoked on a valid source pointer configuration with an assignment $\mathbf{A}$ that is atom-saturated for Π, every unfounded set $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ of Π w.r.t. $\mathbf{A}$ such that all $p \in U$ belong to the same strongly connected component of $DG(\Pi)$ must be contained in S whenever Line 6 of Algorithm 4.3 is entered. If any such unfounded set U is non-empty, this invariant excludes the termination of Algorithm 4.3 by returning $\emptyset$ in Line 18. □

[9] Recall that $\mathbf{A}$ is atom-saturated for Π if $\{p \in atom(\Pi) \mid body_\Pi(p) \subseteq \mathbf{A}^{\boldsymbol{F}}\} \subseteq \mathbf{A}^{\boldsymbol{F}}$.

Theorem 4.7. *Let* Π *be a normal program and* $\mathbf{A}$ *an assignment that is both atom- and body-saturated for* Π.

If UNFOUNDEDSET($\Pi, \mathbf{A}$) *is invoked on a valid source pointer configuration, then we have that* UNFOUNDEDSET($\Pi, \mathbf{A}$) *returns an unfounded set* $U \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$ *of* Π *w.r.t.* $\mathbf{A}$*, where* $U = \emptyset$ *iff there is no unfounded set* U' *of* Π *w.r.t.* $\mathbf{A}$ *such that* $U' \not\subseteq \mathbf{A}^{\boldsymbol{F}}$.

Proof. Assume that UNFOUNDEDSET($\Pi, \mathbf{A}$) is invoked on a valid source pointer configuration. Then, in view of the condition $EB_\Pi(U) \subseteq \mathbf{A}^{\boldsymbol{F}}$ in Line 9 of Algorithm 4.3 and the fact that $\emptyset$, which can be returned in Line 18, is a (trivial) unfounded set of Π w.r.t. $\mathbf{A}$, we have that UNFOUNDEDSET($\Pi, \mathbf{A}$) can only return an unfounded set of Π w.r.t. $\mathbf{A}$. By Corollary 2.10, the existence of some non-empty unfounded set U' of Π w.r.t. $\mathbf{A}$ such that $U' \not\subseteq \mathbf{A}^{\boldsymbol{F}}$ implies that there is a non-empty unfounded set $U \subseteq U' \setminus \mathbf{A}^{\boldsymbol{F}}$ of Π w.r.t. $\mathbf{A}$ such that all $p \in U$ belong to the same strongly connected component of $DG(\Pi)$. Furthermore, by Lemma 4.6, any such unfounded set U of Π w.r.t. $\mathbf{A}$ is contained in the scope S whenever Line 6 is entered. This shows that UNFOUNDEDSET($\Pi, \mathbf{A}$) cannot return $\emptyset$ in Line 18 if there is some non-empty unfounded set U' of Π w.r.t. $\mathbf{A}$ such that $U' \not\subseteq \mathbf{A}^{\boldsymbol{F}}$.

It only remains to show that UNFOUNDEDSET($\Pi, \mathbf{A}$) is terminating. To this end, note that the scope S is increasing over iterations of the loop in Line 2–5 of Algorithm 4.3, and strictly decreasing over iterations of the loop in Line 6–17. For U handled in the loop in Line 8–17, we observe that it is strictly increasing when U is extended in Line 16, and strictly decreasing when an element q is removed from U in Line 14, where q cannot be added back later on because it is also removed from S in Line 15. Since $atom(\Pi)$ is finite and $U \subseteq S \subseteq atom(\Pi) \setminus \mathbf{A}^{\boldsymbol{F}}$, we conclude that none of the loops in Algorithm 4.3 can be iterated infinitely. Rather, any atom can be added to and removed from S and U, respectively, at most once, which yields that the time complexity of UNFOUNDEDSET($\Pi, \mathbf{A}$) is linear in the size of Π. □

Next, we show properties of NOGOODPROPAGATION in Algorithm 4.2 on Page 65. Lemma 4.8 essentially establishes the applicability of Theorem 4.7 whenever an unfounded set check is initiated in Line 12 of Algorithm 4.2, and Lemma 4.9 provides properties crucial for the soundness and completeness of conflict-driven ASP solving.

Lemma 4.8. *Let* Π *be a normal program,* ∇' *a set of nogoods, and* $\mathbf{A}'$ *an assignment.*

Then, we have that $\mathbf{A}$ *is both atom- and body-saturated for* Π *whenever Line 10 of Algorithm 4.2 is entered in an execution of* NOGOODPROPAGATION($\Pi, \nabla', \mathbf{A}'$).

Proof. For the sake of contradiction, assume that Line 10 of Algorithm 4.2 is entered in an execution of NOGOODPROPAGATION($\Pi, \nabla', \mathbf{A}'$), while the current assignment $\mathbf{A}$ is not atom-saturated or not body-saturated for Π. Then, some of the following cases applies:

($body_\Pi(p) \subseteq \mathbf{A}^{\boldsymbol{F}}$ but $\boldsymbol{F}p \notin \mathbf{A}$ for some $p \in atom(\Pi)$) The nogood $\delta = \{\boldsymbol{T}p, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \in \Delta_\Pi$, where $body_\Pi(p) = \{B_1, \ldots, B_k\}$, is such that $\delta \setminus \mathbf{A} \subseteq \{\boldsymbol{T}p\}$. In view of the condition in Line 4 of Algorithm 4.2, tested in the previous iteration of the loop in Line 3–9, we have that $\boldsymbol{T}p \notin \mathbf{A}$ and $\delta \setminus \mathbf{A} = \{\boldsymbol{T}p\}$. But this implies that $\boldsymbol{F}p$ is unit-resulting for δ w.r.t. $\mathbf{A}$, so that the condition $\Sigma = \emptyset$ cannot hold in Line 9, which contradicts that Line 10 of Algorithm 4.2 is entered with $\mathbf{A}$ being the current assignment.

($(B^+ \cap \mathbf{A}^{\boldsymbol{F}}) \cup (B^- \cap \mathbf{A}^{\boldsymbol{T}}) \neq \emptyset$ but $\boldsymbol{F}B \notin \mathbf{A}$ for some $B \in \mathit{body}(\Pi)$) Some nogood $\delta = \{\boldsymbol{T}B, \boldsymbol{f}l\} \in \Delta_\Pi$, where $l \in B$, is such that $\delta \setminus \mathbf{A} \subseteq \{\boldsymbol{T}B\}$. In view of the condition in Line 4 of Algorithm 4.2, tested in the previous iteration of the loop in Line 3–9, we have that $\boldsymbol{T}B \notin \mathbf{A}$ and $\delta \setminus \mathbf{A} = \{\boldsymbol{T}B\}$. But this implies that $\boldsymbol{F}B$ is unit-resulting for δ w.r.t. $\mathbf{A}$, so that the condition $\Sigma = \emptyset$ cannot hold in Line 9, which contradicts that Line 10 of Algorithm 4.2 is entered with $\mathbf{A}$ being the current assignment.

Since each of the above cases yields a contradiction, we conclude that $\mathbf{A}$ is both atom- and body-saturated for Π whenever Line 10 of Algorithm 4.2 is entered. □

Lemma 4.9. *Let Π be a normal program, ∇' a set of nogoods, and $\mathbf{A}'$ an ordered assignment.*

If NOGOODPROPAGATION$(\Pi, \nabla', \mathbf{A}')$ *is invoked on a valid source pointer configuration, then we have that* NOGOODPROPAGATION$(\Pi, \nabla', \mathbf{A}')$ *returns a pair $(\mathbf{A}, \nabla)$ such that*

1. $\nabla' \subseteq \nabla \subseteq \nabla' \cup \Lambda_\Pi$*;*
2. $\mathbf{A}$ *is an ordered assignment such that $\mathbf{A}' \subseteq \mathbf{A}$ and every $\sigma \in \mathbf{A} \setminus \mathbf{A}'$ is implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$;*
3. $\delta \subseteq \mathbf{A}$ *for some $\delta \in \Delta_\Pi \cup \nabla$ if $\varepsilon \subseteq \mathbf{A}$ for some $\varepsilon \in \Lambda_\Pi$.*

Proof. Assume that NOGOODPROPAGATION$(\Pi, \nabla', \mathbf{A}')$ is invoked on a valid source pointer configuration. Then, we start by showing that the items of the statement hold if NOGOODPROPAGATION$(\Pi, \nabla', \mathbf{A}')$ returns a pair $(\mathbf{A}, \nabla)$:

1. Since ∇' can be augmented only with elements of Λ_Π in Line 15 of Algorithm 4.2, we have that $\nabla' \subseteq \nabla \subseteq \nabla' \cup \Lambda_\Pi$.
2. In view of Line 5 of Algorithm 4.2, for each entry σ inserted into an assignment $\mathbf{B}$ such that $\mathbf{A}' \subseteq \mathbf{B} \subset \mathbf{A}$, we have that $\mathit{var}(\sigma) \notin \mathbf{B}^{\boldsymbol{T}} \cup \mathbf{B}^{\boldsymbol{F}}$ and that there is an antecedent $\delta \in \Delta_\Pi \cup \nabla$ of σ w.r.t. $\mathbf{B}$. Since $\mathit{dlevel}(\sigma)$ is set to $\max(\{\mathit{dlevel}(\rho) \mid \rho \in \delta \setminus \{\overline{\sigma}\}\} \cup \{0\})$ in Line 7, we conclude that $\mathbf{B} \circ \sigma$ constructed in Line 8 is an ordered assignment such that σ is implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{B} \circ \sigma$.
3. For the sake of contradiction, assume that $\delta \not\subseteq \mathbf{A}$ for all $\delta \in \Delta_\Pi \cup \nabla$ and $\varepsilon \subseteq \mathbf{A}$ for some $\varepsilon \in \Lambda_\Pi$. Then, there is some unfounded set U of Π w.r.t. $\mathbf{A}$ such that $U \not\subseteq \mathbf{A}^{\boldsymbol{F}}$. Furthermore, $(\mathbf{A}, \nabla)$ must be returned in Line 10 or 13 of Algorithm 4.2, and Lemma 4.8 tells us that $\mathbf{A}$ is both atom- and body-saturated for Π. By Theorem 2.9, we conclude that some $L \in \mathit{loop}(\Pi)$ is unfounded for Π w.r.t. $\mathbf{A}$, so that Π is not tight. Hence, $(\mathbf{A}, \nabla)$ must be returned in Line 13 after obtaining $\emptyset$ as the result of UNFOUNDEDSET$(\Pi, \mathbf{A})$ in Line 12. However, by Lemma 4.8 and 4.5, we have that Theorem 4.7 is applicable, which contradicts that $\emptyset$ is obtained as the result of UNFOUNDEDSET$(\Pi, \mathbf{A})$ in Line 12.

It remains to show that NOGOODPROPAGATION$(\Pi, \nabla', \mathbf{A}')$ is terminating. To this end, note that an assignment $\mathbf{B}$ such that $\mathbf{A}' \subseteq \mathbf{B}$ is increasing over iterations of the loop in Line 3–9 of Algorithm 4.2, as shown in the proof of the second item. Furthermore, by Theorem 4.7 (along with Lemma 4.8 and 4.5), any invocation of UNFOUNDEDSET$(\Pi, \mathbf{B})$

in Line 12 terminates with an unfounded set $U \subseteq atom(\Pi) \setminus \mathbf{B}^{\boldsymbol{F}}$ of Π w.r.t. $\mathbf{B}$. Hence, we have that $U = \emptyset$ or $\{\boldsymbol{T}p, \boldsymbol{F}B_1, \ldots, \boldsymbol{F}B_k\} \setminus \mathbf{B} \subseteq \{\boldsymbol{T}p\}$ for each $p \in U$, where $EB_\Pi(U) = \{B_1, \ldots, B_k\}$; by Lemma 4.8 and Proposition 2.5, the same applies to $U \setminus \mathbf{B}^{\boldsymbol{F}}$ determined in Line 11. Thus, any execution of Line 11–12 is followed by the termination of Algorithm 4.2 in Line 13 or, in view of Line 14–15, by the termination in Line 4 or the insertion of an entry $\boldsymbol{F}p$ (for $p \in atom(\Pi) \setminus (\mathbf{B}^{\boldsymbol{T}} \cup \mathbf{B}^{\boldsymbol{F}})$) into $\mathbf{B}$ in Line 8 in the next iteration of the loop in Line 2–15. Since $atom(\Pi) \cup body(\Pi)$ is finite, there cannot be infinitely many entries added to $\mathbf{A}'$ over iterations of the loops in Line 2–15 and 3–9, respectively, so that NOGOODPROPAGATION$(\Pi, \nabla', \mathbf{A}')$ terminates by returning a pair $(\mathbf{A}, \nabla)$. □

The following lemma, dealing with CONFLICTANALYSIS in Algorithm 4.4 on Page 72, expresses that an asserting nogood is returned when given a nogood violated at an implied decision level greater than zero.

Lemma 4.10. *Let Π be a normal program, ∇ a set of nogoods, $\mathbf{A}$ an ordered assignment, and $\delta' \subseteq \mathbf{A}$ such that $m = \max(\{dlevel(\rho) \mid \rho \in \delta'\} \cup \{0\}) \neq 0$.*

If m is implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$, then we have that CONFLICTANALYSIS$(\delta', \Pi, \nabla, \mathbf{A})$ *returns a pair (δ, k) such that*

1. *$\delta \subseteq \mathbf{A}$;*
2. *$|\{\sigma \in \delta \mid 0 \leq k < dlevel(\sigma)\}| = 1$;*
3. *$\delta \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \nabla \cup \{\delta'\}$.*

Proof. Assume that m is implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$. Since $\mathbf{A}$ is ordered and $m \neq 0$, every $\varepsilon' \subseteq \mathbf{A}$ such that $\max(\{dlevel(\rho) \mid \rho \in \varepsilon'\} \cup \{0\}) = m$ contains an entry σ such that $\varepsilon' \setminus \mathbf{A}[\sigma] = \{\sigma\}$ and $dlevel(\sigma) = m$. (Such entries σ are determined in Line 2 of Algorithm 4.4.) Then, by induction on iterations of the loop in Line 1–7 of Algorithm 4.4, we show that the items of the statement hold if CONFLICTANALYSIS$(\delta', \Pi, \nabla, \mathbf{A})$ returns a pair (δ, k):

(Base case) Let $\delta \subseteq \mathbf{A}$ be some set of entries such that $\max(\{dlevel(\rho) \mid \rho \in \delta\} \cup \{0\}) = m$ and $\delta \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \nabla \cup \{\delta'\}$. For the entry $\sigma \in \delta$ determined in Line 2 of Algorithm 4.4, if the test in Line 4 yields that $k = \max(\{dlevel(\rho) \mid \rho \in \delta \setminus \{\sigma\}\} \cup \{0\}) \neq m$, then $|\{\rho \in \delta \mid 0 \leq k < dlevel(\rho)\}| = |\{\sigma\}| = 1$, and (δ, k) is returned in Line 7.

(Induction step) Let $\varepsilon' \subseteq \mathbf{A}$ be some set of entries such that $\max(\{dlevel(\rho) \mid \rho \in \varepsilon'\} \cup \{0\}) = m$ and $\varepsilon' \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \nabla \cup \{\delta'\}$. For the entry $\sigma \in \varepsilon'$ determined in Line 2 of Algorithm 4.4, if the test in Line 4 yields that $\max(\{dlevel(\rho) \mid \rho \in \varepsilon' \setminus \{\sigma\}\} \cup \{0\}) = m$, there is some $\rho \in \mathbf{A}[\sigma]$ such that $dlevel(\rho) = dlevel(\sigma) = m$. Given that m is implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$, $\Delta_\Pi \cup \nabla$ contains an antecedent of σ w.r.t. $\mathbf{A}$, and some such ε is selected in Line 5. Since $\varepsilon' \setminus \{\sigma\} \subseteq \mathbf{A}[\sigma] \subseteq \mathbf{A}$ and $\varepsilon \setminus \{\overline{\sigma}\} \subseteq \mathbf{A}[\sigma] \subseteq \mathbf{A}$, for $(\varepsilon' \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\})$ constructed in Line 6, it holds that $(\varepsilon' \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\}) \subseteq \mathbf{A}[\sigma] \subseteq \mathbf{A}$. Furthermore, $\max(\{dlevel(\rho) \mid \rho \in \varepsilon' \setminus \{\sigma\}\} \cup \{0\}) = m$ implies that $\max(\{dlevel(\rho) \mid \rho \in (\varepsilon' \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\})\} \cup \{0\}) = m$. Finally, since any solution $\mathbf{B}$ for $\Delta_\Pi \cup \nabla \cup \{\delta'\}$ contains either σ or $\overline{\sigma}$, while $\varepsilon' \not\subseteq \mathbf{B}$ and $\varepsilon \not\subseteq \mathbf{B}$, we have that $(\varepsilon' \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\}) \not\subseteq \mathbf{B}$.

It remains to show that CONFLICTANALYSIS$(\delta', \Pi, \nabla, \mathbf{A})$ is terminating, i.e., that the base case of the induction eventually applies. To this end, note that the prerequisite that $\mathbf{A}$ is ordered implies that $\mathbf{A}$ does not include duplicate entries and must thus be finite. Furthermore, in the induction step above, we have that $(\varepsilon' \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\}) \subseteq \mathbf{A}[\sigma] \subseteq \mathbf{A}$, i.e., all entries $\rho \in (\varepsilon' \setminus \{\sigma\}) \cup (\varepsilon \setminus \{\overline{\sigma}\})$ precede σ in $\mathbf{A}$. From this, we conclude that there cannot be infinitely many applications of the induction step over iterations of the loop in Line 1–7 of Algorithm 4.4, so that CONFLICTANALYSIS$(\delta', \Pi, \nabla, \mathbf{A})$ terminates by returning a pair (δ, k). □

We now turn to CDNL-ASP in Algorithm 4.1 on Page 62 for deciding whether a normal program has an answer set, where Lemma 4.11 establishes invariants that are crucial for the main soundness and completeness result, stated in Theorem 4.12.

Lemma 4.11. *Let Π be a normal program.*

Then, we have that the following holds whenever Line 5 of Algorithm 4.1 is entered in an execution of CDNL-ASP(Π)*:*

1. *∇ is a set of nogoods such that $\delta \not\subseteq \mathbf{B}$ for every $\delta \in \nabla$ and any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$;*
2. *$\mathbf{A}$ is an ordered assignment such that all decision levels are implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$.*

Proof. By induction on iterations of the loop in Line 4–16 of Algorithm 4.1, we show that the items of the statement hold whenever Line 5 is entered in an execution of CDNL-ASP(Π):

(Base case) Before the first iteration, in view of Line 1–3 of Algorithm 4.1, we have that $\mathbf{A} = \emptyset$ and $\nabla = \emptyset$, for which the items of the statement trivially hold, and also that $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}\} \cup \{0\}) \leq dl = 0$.

(Induction step) At the beginning of an iteration of the loop in Line 4–16 of Algorithm 4.1, let ∇' and $\mathbf{A}'$ be such that the items (*1* and *2*) of the statement are satisfied w.r.t. them, and assume that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}'\} \cup \{0\}) \leq dl$.[10] Then, by Lemma 4.9 (along with Lemma 4.5 and 4.8), we have that NOGOOD-PROPAGATION$(\Pi, \nabla', \mathbf{A}')$ invoked in Line 5 returns a pair $(\mathbf{A}, \nabla)$ such that the items of the statement still hold for ∇ and $\mathbf{A}$, respectively. Furthermore, in view of Line 7–8 of Algorithm 4.2, we have that $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}\} \cup \{0\}) \leq dl$. Afterwards, one of the following cases applies w.r.t. ∇ and $\mathbf{A}$:

($\varepsilon \subseteq \mathbf{A}$ for some $\varepsilon \in \Delta_\Pi \cup \nabla$) If the condition in Line 7 of Algorithm 4.1 applies, CDNL-ASP(Π) immediately terminates by returning "no answer set." Otherwise, by Lemma 4.10, CONFLICTANALYSIS$(\varepsilon, \Pi, \nabla, \mathbf{A})$ returns a pair (δ, k) such that $\delta \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \nabla$. Since any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ is a solution for $\Delta_\Pi \cup \nabla$ as well, it follows that $\delta \not\subseteq \mathbf{B}$, so that (*1*) $\mathbf{B}$ is a solution for $\Delta_\Pi \cup (\nabla \cup \{\delta\})$, where $\nabla \cup \{\delta\}$ is constructed in Line 9. Furthermore, $\mathbf{A}_k = \mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid k < dlevel(\sigma)\}$

[10] We below indicate derivations of our induction hypotheses by (*1*) and (*2*), standing for the first and the second item of the statement, respectively, as well as by (*a*), expressing that $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}\} \cup \{0\}) \leq dl$ holds for an assignment $\mathbf{A}$ and the current value of dl.

constructed in Line 10 is an ordered assignment such that (*2*) all decision levels are implied by $\Delta_\Pi \cup (\nabla \cup \{\delta\})$ w.r.t. $\mathbf{A}_k$. Finally, $0 \leq k$ holds in view of the second item in the statement of Lemma 4.10, so that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_k\} \cup \{0\}) \leq k$, where dl is set to k in Line 8. That is, the induction hypotheses still apply w.r.t. $\nabla \cup \{\delta\}$ and $\mathbf{A}_k$.

($\varepsilon \not\subseteq \mathbf{A}$ for all $\varepsilon \in \Delta_\Pi \cup \nabla$) If the condition in Line 11 of Algorithm 4.1 applies, CDNL-ASP(Π) terminates in Line 12 by returning $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$. Otherwise, some entry σ_d such that $var(\sigma_d) \in (atom(\Pi) \cup body(\Pi)) \setminus (\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}})$, as required in Section 4.3.1, is returned by SELECT($\Pi, \nabla, \mathbf{A}$) in Line 14. Let dl be the increment of the former decision level, as set in Line 15. Since $dlevel(\sigma_d)$ is set to dl in Line 15, we have that $\mathbf{A}_{\sigma_d} = \mathbf{A} \circ \sigma_d$ constructed in Line 16 is an ordered assignment such that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\sigma_d}\} \cup \{0\}) = dl$ and (*2*) all decision levels are implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}_{\sigma_d}$. Finally, we note that (*1*) ∇ is not altered. That is, the induction hypotheses still apply w.r.t. ∇ and $\mathbf{A}_{\sigma_d}$.

We have thus shown that the items of the statement hold whenever Line 5 of Algorithm 4.1 is entered. □

Theorem 4.12. *Let* Π *be a normal program.*

Then, we have that CDNL-ASP(Π) *is terminating, and it returns an answer set of* Π *iff* Π *has some answer set.*

Proof. If $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is returned in Line 12 of Algorithm 4.1, then the test in Line 6 and the third item in the statement of Lemma 4.9 establish that $\mathbf{A}$ is a solution for $\Delta_\Pi \cup \Lambda_\Pi$. Furthermore, by Lemma 4.2, we have that there is no solution $\mathbf{B} \neq \mathbf{A}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi) = \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$. Hence, by Theorem 4.4, we conclude that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is an answer set of Π. On the other hand, if CDNL-ASP(Π) returns "no answer set" in Line 7, we have that $\max(\{dlevel(\sigma) \mid \sigma \in \varepsilon\} \cup \{0\}) = 0$ for some $\varepsilon \in \Delta_\Pi \cup \nabla$ such that $\varepsilon \subseteq \mathbf{A}$. By the second item in the statement of Lemma 4.11, for every $\sigma \in \mathbf{A}$ such that $dlevel(\sigma) = 0$, there is some antecedent of σ w.r.t. $\mathbf{A}$ in $\Delta_\Pi \cup \nabla$, so that there cannot be any solution for $\Delta_\Pi \cup \nabla$. Along with the first item in the statement of Lemma 4.11, it follows that there is no solution for $\Delta_\Pi \cup \Lambda_\Pi$. Hence, by Theorem 4.4, we conclude that Π has no answer set.

It remains to show that CDNL-ASP(Π) is terminating. In view of the second item in the statement of Lemma 4.11 and the condition in Line 7 of Algorithm 4.1, we have that Lemma 4.10 applies whenever CONFLICTANALYSIS($\varepsilon, \Pi, \nabla, \mathbf{A}$) is invoked in Line 8. Hence, it returns a pair (δ, k) such that some entry ρ is unit-resulting for δ w.r.t. $\mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid k < dlevel(\sigma)\}$. As a consequence, ρ will be inserted into $\mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid k < dlevel(\sigma)\}$ in Line 5 in the next iteration of the loop in Line 4–16. That is, after every backjump in Line 10, some element of $atom(\Pi) \cup body(\Pi)$ is assigned at a smaller (non-negative) decision level than before. Since $atom(\Pi) \cup body(\Pi)$ is finite, this implies that CDNL-ASP(Π) admits only finitely many backjumps.[11] Along with the fact that

[11] See, e.g., [220, 203] for detailed arguments for the fact that the search pattern combining backjumping with conflict-driven assertions is complete for (UN)SAT. In a nutshell, such arguments work by ranking assignments according to the numbers of variables assigned per decision level and by verifying that the sequence of assignments generated during search is strictly monotonic. Since the total number of variables is finite, every such sequence must be finite as well (yet its length depends on heuristics). Note that this does not necessitate keeping all recorded conflict (or loop) nogoods. Rather, only the antecedents of assigned entries are ultimately needed (for conflict resolution), and their number is bounded by the number of variables.

$\mathbf{A}$ is strictly extended in Line 16, so that either a backjump or termination in Line 12 is inevitable within a linear number of iterations of the loop in Line 4–16, we conclude that CDNL-ASP(Π) eventually terminates in Line 7 or 12 of Algorithm 4.1. □

We have thus proven the formal results presented in Section 4.3.

B.3.3 Section 4.4

In what follows, we show that the enumeration algorithms presented in Section 4.4 are sound, complete, and redundancy-free (w.r.t. "output" atoms) according to Definition 4.1.

To begin with, Lemma 4.13 establishes invariants used to derive the main result in Theorem 4.14 for CDNL-RECORDING in Algorithm 4.5 on Page 78.

Lemma 4.13. *Let Π be a normal program and s an integer.*

Then, we have that the following holds whenever Line 5 of Algorithm 4.5 is entered in an execution of CDNL-RECORDING(Π, s)*:*

1. *∇ is a set of nogoods such that $\delta \not\subseteq \mathbf{B}$ for every $\delta \in \nabla$ and any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi)$ has not yet been printed;*

2. *$\mathbf{A}$ is an ordered assignment such that all decision levels are implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$.*

Proof. By induction on iterations of the loop in Line 4–19 of Algorithm 4.5, we show that the items of the statement hold whenever Line 5 is entered in an execution of CDNL-RECORDING(Π, s):

(Base case) Before the first iteration, in view of Line 1–3 of Algorithm 4.5, we have that $\mathbf{A} = \emptyset$ and $\nabla = \emptyset$, for which the items of the statement trivially hold, and also that $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}\} \cup \{0\}) \leq dl = 0$.

(Induction step) At the beginning of an iteration of the loop in Line 4–19 of Algorithm 4.5, let ∇' and $\mathbf{A}'$ be such that the items (*1* and *2*) of the statement are satisfied w.r.t. them, and assume that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}'\} \cup \{0\}) \leq dl$. (We below indicate derivations of our induction hypotheses by (*1*), (*2*), and (*a*), respectively, as explained in Footnote 10 on Page 161.) Then, by Lemma 4.9 (along with Lemma 4.5 and 4.8), we have that NOGOODPROPAGATION($\Pi, \nabla', \mathbf{A}'$) invoked in Line 5 returns a pair $(\mathbf{A}, \nabla)$ such that the items of the statement still hold for ∇ and $\mathbf{A}$, respectively. Furthermore, in view of Line 7–8 of Algorithm 4.2, we have that $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}\} \cup \{0\}) \leq dl$. Afterwards, one of the following cases applies w.r.t. ∇ and $\mathbf{A}$:

($\varepsilon \subseteq \mathbf{A}$ for some $\varepsilon \in \Delta_\Pi \cup \nabla$) If the condition in Line 7 of Algorithm 4.5 applies, CDNL-RECORDING(Π, s) immediately terminates. Otherwise, by Lemma 4.10, CONFLICTANALYSIS($\varepsilon, \Pi, \nabla, \mathbf{A}$) returns a pair (δ, k) such that $\delta \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \nabla$. Since any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi)$ has not yet been printed is a solution for $\Delta_\Pi \cup \nabla$ as well, it follows that $\delta \not\subseteq \mathbf{B}$, so that (*1*) $\mathbf{B}$ is a solution for $\Delta_\Pi \cup (\nabla \cup \{\delta\})$, where $\nabla \cup \{\delta\}$ is constructed in Line 9. Furthermore, $\mathbf{A}_k = \mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid k < dlevel(\sigma)\}$ constructed in Line 10 is an ordered assignment such that (*2*) all decision levels are implied by $\Delta_\Pi \cup (\nabla \cup \{\delta\})$

w.r.t. $\mathbf{A}_k$. Finally, $0 \leq k$ holds in view of the second item in the statement of Lemma 4.10, so that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_k\} \cup \{0\}) \leq k$, where dl is set to k in Line 8. That is, the induction hypotheses still apply w.r.t. $\nabla \cup \{\delta\}$ and $\mathbf{A}_k$.

($\varepsilon \not\subseteq \mathbf{A}$ for all $\varepsilon \in \Delta_\Pi \cup \nabla$ and $\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}} = atom(\Pi) \cup body(\Pi)$) If the condition in Line 14 of Algorithm 4.5 applies after printing $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ in Line 12, CDNL-RECORDING(Π, s) immediately terminates. Otherwise, by Lemma 4.2, the nogood $\delta = \{\sigma_p \in \mathbf{A} \mid var(\sigma_p) \in atom(\Pi)\}$ is not contained in any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi)$ has not yet been printed, so that (*1*) $\mathbf{B}$ is a solution for $\Delta_\Pi \cup (\nabla \cup \{\delta\})$, where $\nabla \cup \{\delta\}$ is constructed in Line 15. Finally, we note that (*2*) $\mathbf{A}$ and (*a*) dl are not altered. That is, the induction hypotheses still apply w.r.t. $\nabla \cup \{\delta\}$ and $\mathbf{A}$.

($\varepsilon \not\subseteq \mathbf{A}$ for all $\varepsilon \in \Delta_\Pi \cup \nabla$ and $\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}} \neq atom(\Pi) \cup body(\Pi)$) Some entry σ_d such that $var(\sigma_d) \in (atom(\Pi) \cup body(\Pi)) \setminus (\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}})$, as required in Section 4.3.1, is returned by SELECT$(\Pi, \nabla, \mathbf{A})$ in Line 17 of Algorithm 4.5. Let dl be the increment of the former decision level, as set in Line 18. Since $dlevel(\sigma_d)$ is set to dl in Line 18, we have that $\mathbf{A}_{\sigma_d} = \mathbf{A} \circ \sigma_d$ constructed in Line 19 is an ordered assignment such that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\sigma_d}\} \cup \{0\}) = dl$ and (*2*) all decision levels are implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}_{\sigma_d}$. Finally, we note that (*1*) ∇ is not altered. That is, the induction hypotheses still apply w.r.t. ∇ and $\mathbf{A}_{\sigma_d}$.

We have thus shown that the items of the statement hold whenever Line 5 of Algorithm 4.5 is entered. □

Theorem 4.14. *Let Π be a normal program.*

Then, we have that CDNL-RECORDING$(\Pi, 0)$ *is terminating as well as sound, complete, and redundancy-free w.r.t. $atom(\Pi)$.*

Proof. By the same argument as in the proof of Theorem 4.12, we have that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is an answer set of Π if it is printed in Line 12 of Algorithm 4.5, so that CDNL-RECORDING$(\Pi, 0)$ is sound w.r.t. $atom(\Pi)$. In addition, since $\{\sigma_p \in \mathbf{A} \mid var(\sigma_p) \in atom(\Pi)\}$ is persistently recorded in ∇ in Line 15, the condition in Line 6 makes sure that no solution $\mathbf{B}$ for $\Delta_\Pi \cup \nabla$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi) = \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is enumerated later on, so that CDNL-RECORDING$(\Pi, 0)$ is redundancy-free w.r.t. $atom(\Pi)$. On the other hand, if CDNL-RECORDING$(\Pi, 0)$ terminates in Line 7, we have that $\max(\{dlevel(\sigma) \mid \sigma \in \varepsilon\} \cup \{0\}) = 0$ for some $\varepsilon \in \Delta_\Pi \cup \nabla$ such that $\varepsilon \subseteq \mathbf{A}$. By the second item in the statement of Lemma 4.13, for every $\sigma \in \mathbf{A}$ such that $dlevel(\sigma) = 0$, there is some antecedent of σ w.r.t. $\mathbf{A}$ in $\Delta_\Pi \cup \nabla$, so that there cannot be any solution for $\Delta_\Pi \cup \nabla$. Along with the first item in the statement of Lemma 4.13 (and Lemma 4.2), it follows that all solutions for $\Delta_\Pi \cup \Lambda_\Pi$ have been enumerated. By Theorem 4.4, for every answer set X of Π, the enumerated solutions for $\Delta_\Pi \cup \Lambda_\Pi$ include some $\mathbf{B}$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi) = X$. Hence, from the property that CDNL-RECORDING$(\Pi, 0)$ is terminating, we can conclude that it is complete w.r.t. $atom(\Pi)$.

It remains to show that CDNL-RECORDING$(\Pi, 0)$ is terminating. By the same argument as in the proof of Theorem 4.12, we have that CDNL-RECORDING$(\Pi, 0)$ admits only finitely many backjumps in Line 10 of Algorithm 4.5. Furthermore, when a solution $\mathbf{B}$ for $\Delta_\Pi \cup \nabla$ is enumerated, the persistent recording of $\{\sigma_p \in \mathbf{B} \mid var(\sigma_p) \in$

$atom(\Pi)\}$ in Line 15 leads to either termination in Line 7 or a backjump in Line 10 in the next iteration of the loop in Line 4–19. Along with the fact that $\mathbf{A}$ is strictly extended in Line 19, so that either a backjump or a solution for $\Delta_\Pi \cup \nabla$ is inevitable within a linear number of iterations of the loop in Line 4–19, we conclude that CDNL-RECORDING$(\Pi, 0)$ eventually terminates in Line 7 of Algorithm 4.5. □

We now turn to CDNL-ENUMERATION in Algorithm 4.6 on Page 81, where Lemma 4.15 establishes invariants used to derive the main result in Theorem 4.16.

Lemma 4.15. *Let Π be a normal program and s an integer.*

Then, we have that the following holds whenever Line 5 of Algorithm 4.6 is entered in an execution of CDNL-ENUMERATION(Π, s)*:*

1. *∇ is a set of nogoods such that $\delta \not\subseteq \mathbf{B}$ for every $\delta \in \nabla$ and any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$;*

2. *$\mathbf{A}$ is an ordered assignment such that all decision levels greater than bl are implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$;*

3. *if $\mathbf{A} \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi)$ has not yet been printed, then $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\}$ and $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\}) \notin \mathbf{B}$.*

Proof. By induction on iterations of the loop in Line 4–30 of Algorithm 4.6, we show that the items of the statement hold whenever Line 5 is entered in an execution of CDNL-ENUMERATION(Π, s):

(Base case) Before the first iteration, in view of Line 1–3 of Algorithm 4.6, we have that $\mathbf{A} = \emptyset$ and $\nabla = \emptyset$, for which the items of the statement trivially hold, and also that $0 = bl \leq \max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}\} \cup \{0\}) \leq dl = 0$.

(Induction step) At the beginning of an iteration of the loop in Line 4–30 of Algorithm 4.6, let ∇' and $\mathbf{A}'$ be such that the items (*1*, *2*, and *3*) of the statement are satisfied w.r.t. them, and assume that (*a*) $bl \leq \max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}'\} \cup \{0\}) \leq dl$ and (*b*) $decision(i) \in \mathbf{A}'$ for all $1 \leq i \leq dl$.[12] Then, by Lemma 4.9 (along with Lemma 4.5 and 4.8), we have that NOGOODPROPAGATION$(\Pi, \nabla', \mathbf{A}')$ invoked in Line 5 returns a pair $(\mathbf{A}, \nabla)$ such that the items of the statement still hold for ∇ and $\mathbf{A}$, respectively. In particular, for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi)$ has not yet been printed and any $\sigma \in \mathbf{A} \setminus (\mathbf{A}' \cup \mathbf{B})$, there is some antecedent $\delta \in \Delta_\Pi \cup \nabla$ of σ w.r.t. $\mathbf{A}$, which implies that $\mathbf{A}[\sigma] \not\subseteq \mathbf{B}$ because $\overline{\sigma} \in \mathbf{B}$ yet $\delta \not\subseteq \mathbf{B}$. Hence, if $\mathbf{A} \not\subseteq \mathbf{B}$, we conclude that $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\} = \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}' \setminus \mathbf{B}\}$, so that $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\}) \notin \mathbf{B}$. Furthermore, in view of Line 7–8 of Algorithm 4.2, we have that $bl \leq \max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}\} \cup \{0\}) \leq dl$. Afterwards, one of the following cases applies w.r.t. ∇ and $\mathbf{A}$:

[12] We below indicate derivations of our induction hypotheses by (*1*), (*2*), and (*3*), standing for the first, the second, and the third item of the statement, respectively, as well as by (*a*) and (*b*), expressing that $bl \leq \max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}\} \cup \{0\}) \leq dl$ or, respectively, that $decision(i) \in \mathbf{A}$ for all $1 \leq i \leq dl$ holds for an assignment $\mathbf{A}$ and the current values of bl and dl.

($\varepsilon \subseteq \mathbf{A}$ for some $\varepsilon \in \Delta_\Pi \cup \nabla$) If the condition in Line 7 of Algorithm 4.6 applies, CDNL-ENUMERATION(Π, s) immediately terminates. Otherwise, one of the following subcases applies:

- ($bl < \max\{dlevel(\sigma) \mid \sigma \in \varepsilon\}$) By Lemma 4.10, CONFLICT-ANALYSIS($\varepsilon, \Pi, \nabla, \mathbf{A}$) returns a pair (δ, k) such that $\delta \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \nabla$. Since any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ is a solution for $\Delta_\Pi \cup \nabla$ as well, it follows that $\delta \not\subseteq \mathbf{B}$, so that (*1*) $\mathbf{B}$ is a solution for $\Delta_\Pi \cup (\nabla \cup \{\delta\})$, where $\nabla \cup \{\delta\}$ is constructed in Line 10 of Algorithm 4.6. Furthermore, $\mathbf{A}_{\max} = \mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid \max\{k, bl\} < dlevel(\sigma)\}$ constructed in Line 12 is an ordered assignment such that (*2*) all decision levels greater than bl are implied by $\Delta_\Pi \cup (\nabla \cup \{\delta\})$ w.r.t. $\mathbf{A}_{\max}$. In addition, if $\mathbf{A}_{\max} \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi)$ has not yet been printed, then (*3*) $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\max} \setminus \mathbf{B}\}$ and $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\max} \setminus \mathbf{B}\}) \notin \mathbf{B}$. Finally, $0 \leq k < dl$ holds in view of the first two items in the statement of Lemma 4.10, so that (*b*) $decision(i) \in \mathbf{A}_{\max}$ for all $1 \leq i \leq \max\{k, bl\}$, where dl is set to $\max\{k, bl\}$ in Line 11, and (*a*) $bl \leq \max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\max}\} \cup \{0\}) \leq \max\{k, bl\}$. That is, the induction hypotheses still apply w.r.t. $\nabla \cup \{\delta\}$ and $\mathbf{A}_{\max}$.
- ($\max\{dlevel(\sigma) \mid \sigma \in \varepsilon\} \leq bl$) In view of Line 28–29 of Algorithm 4.6, we have that $dlevel(\sigma_d) = bl$ for $\sigma_d = decision(bl)$ determined in Line 14, so that $\mathbf{A}_{\overline{\sigma_d}} = (\mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid bl - 1 < dlevel(\sigma)\}) \circ \overline{\sigma_d}$ constructed in Line 16–17 is an ordered assignment. As $dlevel(\overline{\sigma_d})$, bl, and dl are set to the decrement of bl in Line 15, it also holds that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\overline{\sigma_d}}\} \cup \{0\}) = bl - 1$, (*b*) $decision(i) \in \mathbf{A}_{\overline{\sigma_d}}$ for all $1 \leq i \leq bl-1$, and (*2*) all decision levels greater than $bl-1$ are (trivially) implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}_{\overline{\sigma_d}}$. Furthermore, for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi)$ has not yet been printed, $\varepsilon \not\subseteq \mathbf{B}$ yields that $\mathbf{A} \not\subseteq \mathbf{B}$, $0 < m = \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\} \leq \max\{dlevel(\sigma) \mid \sigma \in \varepsilon\} \leq bl$, and $decision(m) \notin \mathbf{B}$. Hence, if $m = bl$, we have that $\sigma_d \notin \mathbf{B}$, $\overline{\sigma_d} \in \mathbf{B}$, and $\mathbf{A}_{\overline{\sigma_d}} \subseteq \mathbf{B}$, while (*3*) $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\overline{\sigma_d}} \setminus \mathbf{B}\} = m$ and $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\overline{\sigma_d}} \setminus \mathbf{B}\}) \notin \mathbf{B}$ hold otherwise. Finally, we note that (*1*) ∇ is not altered. That is, the induction hypotheses still apply w.r.t. ∇ and $\mathbf{A}_{\overline{\sigma_d}}$.

($\varepsilon \not\subseteq \mathbf{A}$ for all $\varepsilon \in \Delta_\Pi \cup \nabla$ and $\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}} = atom(\Pi) \cup body(\Pi)$) If the condition in Line 21 of Algorithm 4.6 applies after printing $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ in Line 19, CDNL-ENUMERATION(Π, s) immediately terminates. Otherwise, in view of Line 28–29, we have that $dlevel(\sigma_d) = dl$ for $\sigma_d = decision(dl)$ determined in Line 22, so that $\mathbf{A}_{\overline{\sigma_d}} = (\mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid dl - 1 < dlevel(\sigma)\}) \circ \overline{\sigma_d}$ constructed in Line 24–25 is an ordered assignment. As $dlevel(\overline{\sigma_d})$, bl, and dl are set to the decrement of dl in Line 23, it also holds that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\overline{\sigma_d}}\} \cup \{0\}) = dl - 1$, (*b*) $decision(i) \in \mathbf{A}_{\overline{\sigma_d}}$ for all $1 \leq i \leq dl - 1$, and (*2*) all decision levels greater than $dl - 1$ are (trivially) implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}_{\overline{\sigma_d}}$. Furthermore, for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi)$ has not yet been printed, $\mathbf{A} \not\subseteq \mathbf{B}$ yields that $0 < m = \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\} \leq dl$ and

$decision(m) \notin \mathbf{B}$. Hence, if $m = dl$, we have that $\sigma_d \notin \mathbf{B}$, $\overline{\sigma_d} \in \mathbf{B}$, and $\mathbf{A}_{\overline{\sigma_d}} \subseteq \mathbf{B}$, while (*3*) $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\overline{\sigma_d}} \setminus \mathbf{B}\} = m$ and $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\overline{\sigma_d}} \setminus \mathbf{B}\}) \notin \mathbf{B}$ hold otherwise. Finally, we note that (*1*) ∇ is not altered. That is, the induction hypotheses still apply w.r.t. ∇ and $\mathbf{A}_{\overline{\sigma_d}}$.

($\varepsilon \not\subseteq \mathbf{A}$ for all $\varepsilon \in \Delta_\Pi \cup \nabla$ and $\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}} \neq atom(\Pi) \cup body(\Pi)$) Some entry σ_d such that $var(\sigma_d) \in (atom(\Pi) \cup body(\Pi)) \setminus (\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}})$, as required in Section 4.3.1, is returned by SELECT$(\Pi, \nabla, \mathbf{A})$ in Line 27 of Algorithm 4.6. Let dl be the increment of the former decision level, as set in Line 28. Since $dlevel(\sigma_d)$ is set to dl in Line 28, we have that $\mathbf{A}_{\sigma_d} = \mathbf{A} \circ \sigma_d$ constructed in Line 30 is an ordered assignment such that (*a*) $bl < \max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\sigma_d}\} \cup \{0\}) = dl$ and (*2*) all decision levels greater than bl are implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}_{\sigma_d}$. As $decision(dl)$ is set to σ_d in Line 29, we also have that (*b*) $decision(i) \in \mathbf{A}_{\sigma_d}$ for all $1 \leq i \leq dl$. Furthermore, if $\mathbf{A}_{\sigma_d} \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi)$ has not yet been printed, then (*3*) $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\sigma_d} \setminus \mathbf{B}\}$ and $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\sigma_d} \setminus \mathbf{B}\}) \notin \mathbf{B}$. Finally, we note that (*1*) ∇ is not altered. That is, the induction hypotheses still apply w.r.t. ∇ and $\mathbf{A}_{\sigma_d}$.

We have thus shown that the items of the statement hold whenever Line 5 of Algorithm 4.6 is entered. □

Theorem 4.16. *Let* Π *be a normal program.*

Then, we have that CDNL-ENUMERATION$(\Pi, 0)$ *is terminating as well as sound, complete, and redundancy-free w.r.t.* $atom(\Pi)$.

Proof. By the same argument as in the proof of Theorem 4.12, we have that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is an answer set of Π if it is printed in Line 19 of Algorithm 4.6, so that CDNL-ENUMERATION$(\Pi, 0)$ is sound w.r.t. $atom(\Pi)$. For an enumerated solution $\mathbf{A}$ for $\Delta_\Pi \cup \Lambda_\Pi$, entries of $\mathbf{A}$ can be retracted only in Line 16 or 24, where the complement $\overline{\sigma_d}$ of a former decision entry $\sigma_d \in \mathbf{A}$ is assigned immediately afterwards, that is, in Line 17 or 25, respectively. Hence, any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ enumerated after $\mathbf{A}$ satisfies $\mathbf{B} \neq \mathbf{A}$. By Lemma 4.2, this implies that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi) \neq \mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$, so that CDNL-ENUMERATION$(\Pi, 0)$ is redundancy-free w.r.t. $atom(\Pi)$. On the other hand, by the first and the third item in the statement of Lemma 4.15 (along with Lemma 4.2), we have that neither of the termination conditions in Line 7 and 21 can apply as long as there is a not yet enumerated solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$.[13] By Theorem 4.4, for every answer set X of Π, the solutions for $\Delta_\Pi \cup \Lambda_\Pi$ include some $\mathbf{B}$ such that $\mathbf{B}^{\boldsymbol{T}} \cap atom(\Pi) = X$. Hence, from the property that CDNL-ENUMERATION$(\Pi, 0)$ is terminating, we can conclude that it is complete w.r.t. $atom(\Pi)$.

It remains to show that CDNL-ENUMERATION$(\Pi, 0)$ is terminating. In view of the second item in the statement of Lemma 4.15 and the conditions in Line 7 and 8 of Algorithm 4.6, we have that Lemma 4.10 applies whenever CONFLICTANALYSIS$(\varepsilon, \Pi, \nabla, \mathbf{A})$ is invoked in Line 9. Hence, it returns a pair (δ, k) such that some entry ρ is unit-resulting

[13] Regarding the condition $dl = 0$, tested in Line 21, the fact that $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}\} \cup \{0\}) \leq dl$ has been demonstrated in the proof of Lemma 4.15.

for δ w.r.t. $\mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid \max\{k, bl\} < dlevel(\sigma)\}$.[14] As a consequence, ρ will be inserted into $\mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid \max\{k, bl\} < dlevel(\sigma)\}$ in Line 5 in the next iteration of the loop in Line 4–30. That is, after every backjump in Line 12, some element of $atom(\Pi) \cup body(\Pi)$ is assigned at a smaller (non-negative) decision level than before. Similarly, if backtracking takes place in Line 16 or 24, the complement $\overline{\sigma_d}$ of a former decision entry $\sigma_d \in \mathbf{A}$ is assigned immediately afterwards, that is, in Line 17 or 25, respectively, and $dlevel(\overline{\sigma_d}) = dlevel(\sigma_d) - 1$ holds in view of Line 28–29, 14–15, and 22–23. Since $atom(\Pi) \cup body(\Pi)$ is finite, this implies that CDNL-ENUMERATION$(\Pi, 0)$ admits only finitely many backjumps or backtracks, respectively. Along with the fact that $\mathbf{A}$ is strictly extended in Line 30, so that either a backjump (in Line 12) or a backtrack (in Line 24) is inevitable within a linear number of iterations of the loop in Line 4–30, we conclude that CDNL-ENUMERATION$(\Pi, 0)$ eventually terminates in Line 7 or 21 of Algorithm 4.6. □

Finally, we investigate CDNL-PROJECTION in Algorithm 4.7 on Page 87, where Lemma 4.17 establishes invariants used to derive the main result in Theorem 4.18.

Lemma 4.17. *Let Π be a normal program, P a set of atoms, and s an integer.*

Then, we have that the following holds whenever Line 5 of Algorithm 4.7 is entered in an execution of CDNL-PROJECTION(Π, P, s)*:*

1. *∇ is a set of nogoods such that $\delta \not\subseteq \mathbf{B}$ for every $\delta \in \nabla$ and any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed;*

2. *$\mathbf{A}$ is an ordered assignment such that all decision levels greater than bl are implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}$;*

3. *if $\mathbf{A} \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed, then $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\}$;*

4. *if $\mathbf{A} \not\subseteq \mathbf{B}$ and $\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\} \leq bl$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed, then $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\}) \in (\mathbf{A} \setminus \mathbf{B})^P$.*

Proof. By induction on iterations of the loop in Line 4–41 of Algorithm 4.7, we show that the items of the statement hold whenever Line 5 is entered in an execution of CDNL-PROJECTION(Π, P, s):

(Base case) Before the first iteration, in view of Line 1–3 of Algorithm 4.7, we have that $\mathbf{A} = \emptyset$ and $\nabla = \emptyset$, for which the items of the statement trivially hold, and also that $0 = bl \leq \max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}\} \cup \{0\}) \leq dl = 0$.

(Induction step) At the beginning of an iteration of the loop in Line 4–41 of Algorithm 4.7, let ∇' and $\mathbf{A}'$ be such that the items (*1*, *2*, *3*, and *4*) of the statement are satisfied w.r.t. them, and assume that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}'\} \cup \{0\}) \leq dl$, (*b*) $decision(i) \in (\mathbf{A}')^P$ for all $1 \leq i \leq bl$, and (*c*) $nogood(i) \in \nabla'$ for all

[14] Regarding $bl < \max\{dlevel(\sigma) \mid \sigma \in \delta\}$, the fact that $\max\{dlevel(\sigma) \mid \sigma \in \delta\} = \max\{dlevel(\sigma) \mid \sigma \in \varepsilon\}$ has been demonstrated in the proof of Lemma 4.10.

$1 \leq i \leq bl$.[15] Then, by Lemma 4.9 (along with Lemma 4.5 and 4.8), we have that NOGOODPROPAGATION$(\Pi, \nabla', \mathbf{A}')$ invoked in Line 5 returns a pair $(\mathbf{A}, \nabla)$ such that the items of the statement still hold for ∇ and $\mathbf{A}$, respectively. In particular, for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed and any $\sigma \in \mathbf{A} \setminus (\mathbf{A}' \cup \mathbf{B})$, there is some antecedent $\delta \in \Delta_\Pi \cup \nabla$ of σ w.r.t. $\mathbf{A}$, which implies that $\mathbf{A}[\sigma] \not\subseteq \mathbf{B}$ because $\overline{\sigma} \in \mathbf{B}$ yet $\delta \not\subseteq \mathbf{B}$. Hence, if $\mathbf{A} \not\subseteq \mathbf{B}$, we conclude that $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\} = \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}' \setminus \mathbf{B}\}$. Furthermore, in view of Line 7–8 of Algorithm 4.2, we have that $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}\} \cup \{0\}) \leq dl$. Afterwards, one of the following cases applies w.r.t. ∇ and $\mathbf{A}$:

($\varepsilon \subseteq \mathbf{A}$ for some $\varepsilon \in \Delta_\Pi \cup \nabla$) If the condition in Line 7 of Algorithm 4.7 applies, CDNL-PROJECTION(Π, P, s) immediately terminates. Otherwise, one of the following subcases applies:

- **($bl < \max\{dlevel(\sigma) \mid \sigma \in \varepsilon\}$)** By Lemma 4.10, CONFLICT-ANALYSIS$(\varepsilon, \Pi, \nabla, \mathbf{A})$ returns a pair (δ, k) such that $\delta \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \nabla$. Since any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed is a solution for $\Delta_\Pi \cup \nabla$ as well, it follows that $\delta \not\subseteq \mathbf{B}$, so that (*1*) $\mathbf{B}$ is a solution for $\Delta_\Pi \cup (\nabla \cup \{\delta\})$, where $\nabla \cup \{\delta\}$ is constructed in Line 10 of Algorithm 4.7. Furthermore, $\mathbf{A}_{\max} = \mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid \max\{k, bl\} < dlevel(\sigma)\}$ constructed in Line 12 is an ordered assignment such that (*2*) all decision levels greater than bl are implied by $\Delta_\Pi \cup (\nabla \cup \{\delta\})$ w.r.t. $\mathbf{A}_{\max}$. In addition, if $\mathbf{A}_{\max} \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed, then (*3*) $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\max} \setminus \mathbf{B}\}$ and (*4*) $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\max} \setminus \mathbf{B}\}) \in (\mathbf{A}_{\max} \setminus \mathbf{B})^P$ if $\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\max} \setminus \mathbf{B}\} \leq bl \leq \max\{k, bl\}$. Also, for all $1 \leq i \leq bl$, we have that (*b*) $decision(i) \in \mathbf{A}^P_{\max}$ and (*c*) $nogood(i) \in \nabla \cup \{\delta\}$. Finally, $0 \leq k$ holds in view of the second item in the statement of Lemma 4.10, so that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\max}\} \cup \{0\}) \leq \max\{k, bl\}$, where dl is set to $\max\{k, bl\}$ in Line 11. That is, the induction hypotheses still apply w.r.t. $\nabla \cup \{\delta\}$ and $\mathbf{A}_{\max}$.
- **($\max\{dlevel(\sigma) \mid \sigma \in \varepsilon\} \leq bl$)** In view of Line 35–36 of Algorithm 4.7, we have that $dlevel(\sigma_d) = bl$ for $\sigma_d = decision(bl)$ determined in Line 15, so that $\mathbf{A}_{\overline{\sigma_d}} = (\mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid bl - 1 < dlevel(\sigma)\}) \circ \overline{\sigma_d}$ constructed in Line 17–18 is an ordered assignment. As $dlevel(\overline{\sigma_d})$, bl, and dl are set to the decrement of bl in Line 16, it also holds that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\overline{\sigma_d}}\} \cup \{0\}) = bl - 1$ and (*b*) $decision(i) \in \mathbf{A}^P_{\overline{\sigma_d}}$ for all $1 \leq i \leq bl - 1$. In addition, (*c*) $nogood(i) \in \nabla \setminus \{nogood(bl)\}$ for all $1 \leq i \leq bl - 1$,[16] and (*2*) all decision levels greater than $bl - 1$ are (trivially) implied by $\Delta_\Pi \cup (\nabla \setminus \{nogood(bl)\})$ w.r.t. $\mathbf{A}_{\overline{\sigma_d}}$, where $\nabla \setminus \{nogood(bl)\}$ is constructed in Line 14. Furthermore, for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$

[15] We below indicate derivations of our induction hypotheses by (*1*), (*2*), (*3*), and (*4*), standing for the first, the second, the third, and the fourth item of the statement, respectively, as well as by (*a*), (*b*), and (*c*), expressing that $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}\} \cup \{0\}) \leq dl$, $decision(i) \in \mathbf{A}^P$ for all $1 \leq i \leq bl$, or, respectively, that $nogood(i) \in \nabla$ for all $1 \leq i \leq bl$ holds for an assignment $\mathbf{A}$, a set ∇ of nogoods, and the current values of bl and dl.

[16] For every $1 \leq i \leq bl - 1$, $nogood(i) \neq nogood(bl)$ holds in view of the condition in Line 6, tested before $nogood(bl)$ can be introduced in Line 32.

such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed, $\varepsilon \not\subseteq \mathbf{B}$ yields that $\mathbf{A} \not\subseteq \mathbf{B}$, $0 < m = \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\} \leq \max\{dlevel(\sigma) \mid \sigma \in \varepsilon\} \leq bl$, and $decision(m) \in (\mathbf{A} \setminus \mathbf{B})^P$. Hence, if $m = bl$, we have that $\sigma_d \notin \mathbf{B}$, $\overline{\sigma_d} \in \mathbf{B}$, and $\mathbf{A}_{\overline{\sigma_d}} \subseteq \mathbf{B}$, while (*3*) $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\overline{\sigma_d}} \setminus \mathbf{B}\} = m$ and (*4*) $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\overline{\sigma_d}} \setminus \mathbf{B}\}) \in (\mathbf{A}_{\overline{\sigma_d}} \setminus \mathbf{B})^P$ hold otherwise. Finally, we note that (*1*) $\delta \not\subseteq \mathbf{B}$ for every $\delta \in \nabla \setminus \{nogood(bl)\}$. That is, the induction hypotheses still apply w.r.t. $\nabla \setminus \{nogood(bl)\}$ and $\mathbf{A}_{\overline{\sigma_d}}$.

($\varepsilon \not\subseteq \mathbf{A}$ for all $\varepsilon \in \Delta_\Pi \cup \nabla$ and $\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}} = atom(\Pi) \cup body(\Pi)$) If the condition in Line 22 of Algorithm 4.7 applies after printing $\mathbf{A}^{\boldsymbol{T}} \cap P$ in Line 20, CDNL-PROJECTION(Π, P, s) immediately terminates. Otherwise, one of the following subcases applies:

($\max\{dlevel(\sigma_p) \mid \sigma_p \in \mathbf{A}^P\} = bl$) In view of Line 35–36 of Algorithm 4.7, we have that $dlevel(\sigma_d) = bl$ for $\sigma_d = decision(bl)$ determined in Line 25, so that $\mathbf{A}_{\overline{\sigma_d}} = (\mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid bl-1 < dlevel(\sigma)\}) \circ \overline{\sigma_d}$ constructed in Line 27–28 is an ordered assignment. As $dlevel(\overline{\sigma_d})$, bl, and dl are set to the decrement of bl in Line 26, it also holds that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\overline{\sigma_d}}\} \cup \{0\}) = bl-1$ and (*b*) $decision(i) \in \mathbf{A}^P_{\overline{\sigma_d}}$ for all $1 \leq i \leq bl-1$. In addition, (*c*) $nogood(i) \in \nabla \setminus \{nogood(bl)\}$ for all $1 \leq i \leq bl-1$, and (*2*) all decision levels greater than $bl-1$ are (trivially) implied by $\Delta_\Pi \cup (\nabla \setminus \{nogood(bl)\})$ w.r.t. $\mathbf{A}_{\overline{\sigma_d}}$, where $\nabla \setminus \{nogood(bl)\}$ is constructed in Line 24. Furthermore, for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed, $\mathbf{A}^P \not\subseteq \mathbf{B}$ yields that $0 < m = \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A} \setminus \mathbf{B}\} \leq bl$ and $decision(m) \in (\mathbf{A} \setminus \mathbf{B})^P$. Hence, if $m = bl$, we have that $\sigma_d \notin \mathbf{B}$, $\overline{\sigma_d} \in \mathbf{B}$, and $\mathbf{A}_{\overline{\sigma_d}} \subseteq \mathbf{B}$, while (*3*) $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\overline{\sigma_d}} \setminus \mathbf{B}\} = m$ and (*4*) $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\overline{\sigma_d}} \setminus \mathbf{B}\}) \in (\mathbf{A}_{\overline{\sigma_d}} \setminus \mathbf{B})^P$ hold otherwise. Finally, we note that (*1*) $\delta \not\subseteq \mathbf{B}$ for every $\delta \in \nabla \setminus \{nogood(bl)\}$. That is, the induction hypotheses still apply w.r.t. $\nabla \setminus \{nogood(bl)\}$ and $\mathbf{A}_{\overline{\sigma_d}}$.

($\max\{dlevel(\sigma_p) \mid \sigma_p \in \mathbf{A}^P\} \neq bl$) As $nogood(bl+1)$ is set to $\mathbf{A}^P$ in Line 31 of Algorithm 4.7, we have that (*c*) $nogood(i) \in \nabla \cup \{nogood(bl+1)\}$ for all $1 \leq i \leq bl+1$, where bl is incremented in Line 30 and $\nabla \cup \{nogood(bl+1)\}$ is constructed in Line 32. Given that the conditions in Line 7–8 and 22–23, tested before bl can be decremented in Line 16 or 26, respectively, make sure that bl is non-negative, along with $decision(i) \in \mathbf{A}^P$ and $dlevel(decision(i)) = i$ (in view of Line 35–36) for all $1 \leq i \leq bl$, it holds that $0 \leq bl < \max\{dlevel(\sigma_p) \mid \sigma_p \in \mathbf{A}^P\}$. Since some entry $\sigma_d \in nogood(bl+1)$ such that $bl < dlevel(\sigma_d)$ is selected in Line 34 and $dlevel(\sigma_d)$ as well as dl are set to the increment of bl in Line 35, $\mathbf{A}_{\sigma_d} = (\mathbf{A} \setminus \{\sigma \in \mathbf{A} \mid bl < dlevel(\sigma)\}) \circ \sigma_d$ constructed in Line 33 and 37 is an ordered assignment such that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\sigma_d}\} \cup \{0\}) = bl+1$ and (*2*) all decision levels greater than $bl+1$ are (trivially) implied by $\Delta_\Pi \cup (\nabla \cup \{nogood(bl+1)\})$ w.r.t. $\mathbf{A}_{\sigma_d}$. Furthermore, as $decision(bl+1)$ is set to σ_d in Line 36, we have that (*b*) $decision(i) \in \mathbf{A}^P_{\sigma_d}$ for all $1 \leq i \leq bl+1$. For any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed, since

$\mathbf{A}^P \not\subseteq \mathbf{B}$, it holds that (*1*) $\delta \not\subseteq \mathbf{B}$ for every $\delta \in \nabla \cup \{nogood(bl+1)\}$. Finally, if $\mathbf{A}_{\sigma_d} \not\subseteq \mathbf{B}$, then (*3*) $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\sigma_d} \setminus \mathbf{B}\}$ and (*4*) $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\sigma_d} \setminus \mathbf{B}\}) \in (\mathbf{A}_{\sigma_d} \setminus \mathbf{B})^P$. That is, the induction hypotheses still apply w.r.t. $\nabla \cup \{nogood(bl+1)\}$ and $\mathbf{A}_{\sigma_d}$.

($\varepsilon \not\subseteq \mathbf{A}$ for all $\varepsilon \in \Delta_\Pi \cup \nabla$ and $\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}} \neq atom(\Pi) \cup body(\Pi)$) Some entry σ_d such that $var(\sigma_d) \in (atom(\Pi) \cup body(\Pi)) \setminus (\mathbf{A}^{\boldsymbol{T}} \cup \mathbf{A}^{\boldsymbol{F}})$, as required in Section 4.3.1, is returned by SELECT$(\Pi, \nabla, \mathbf{A})$ in Line 39 of Algorithm 4.7. Let dl be the increment of the former decision level, as set in Line 40. Since $dlevel(\sigma_d)$ is set to dl in Line 40, we have that $\mathbf{A}_{\sigma_d} = \mathbf{A} \circ \sigma_d$ constructed in Line 41 is an ordered assignment such that (*a*) $\max(\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\sigma_d}\} \cup \{0\}) = dl$ and (*2*) all decision levels greater than bl are implied by $\Delta_\Pi \cup \nabla$ w.r.t. $\mathbf{A}_{\sigma_d}$. As bl and (*1*) ∇ are not altered, for all $1 \leq i \leq bl < dl$, we have that (*b*) $decision(i) \in \mathbf{A}^P_{\sigma_d}$ and (*c*) $nogood(i) \in \nabla$. Finally, if $\mathbf{A}_{\sigma_d} \not\subseteq \mathbf{B}$ for any solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed, then (*3*) $0 < \min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\sigma_d} \setminus \mathbf{B}\}$ and (*4*) $decision(\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\sigma_d} \setminus \mathbf{B}\}) \in (\mathbf{A}_{\sigma_d} \setminus \mathbf{B})^P$ if $\min\{dlevel(\sigma) \mid \sigma \in \mathbf{A}_{\sigma_d} \setminus \mathbf{B}\} \leq bl$. That is, the induction hypotheses still apply w.r.t. ∇ and $\mathbf{A}_{\sigma_d}$.

We have thus shown that the items of the statement hold whenever Line 5 of Algorithm 4.7 is entered. □

Theorem 4.18. *Let Π be a normal program and P a set of atoms.*

Then, we have that CDNL-PROJECTION$(\Pi, P, 0)$ *is terminating as well as sound, complete, and redundancy-free w.r.t. P.*

Proof. By the same argument as in the proof of Theorem 4.12, we have that $\mathbf{A}^{\boldsymbol{T}} \cap atom(\Pi)$ is an answer set of Π if $\mathbf{A}^{\boldsymbol{T}} \cap P$ is printed in Line 20 of Algorithm 4.7, so that CDNL-PROJECTION$(\Pi, P, 0)$ is sound w.r.t. P. For an enumerated solution $\mathbf{A}$ for $\Delta_\Pi \cup \Lambda_\Pi$, (fake) decision entries $\sigma_d \in \mathbf{A}^P$ such that $dlevel(\sigma_d) \leq bl$ (bl and σ_d may be obtained by incrementation in Line 30 along with reassignment in Line 37) can be retracted only in Line 17 or 27, where the complement $\overline{\sigma_d}$ of σ_d is assigned immediately afterwards, that is, in Line 18 or 28, respectively. Until then, $nogood(bl) = \mathbf{A}^P$, (possibly) recorded in ∇ in Line 32 (and deleted from ∇ in Line 14 or 24, respectively), excludes all assignments $\mathbf{B}$ such that $\mathbf{A}^P \subseteq \mathbf{B}$ as solutions for $\Delta_\Pi \cup \nabla$, so that CDNL-PROJECTION$(\Pi, P, 0)$ is redundancy-free w.r.t. P. On the other hand, by the first and the third item in the statement of Lemma 4.17, we have that neither of the termination conditions in Line 7 and 22 can apply as long as there is a solution $\mathbf{B}$ for $\Delta_\Pi \cup \Lambda_\Pi$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P$ has not yet been printed. By Theorem 4.4, for every answer set X of Π, the solutions for $\Delta_\Pi \cup \Lambda_\Pi$ include some $\mathbf{B}$ such that $\mathbf{B}^{\boldsymbol{T}} \cap P = X \cap P$. Hence, from the property that CDNL-PROJECTION$(\Pi, P, 0)$ is terminating, we can conclude that it is complete w.r.t. P.

It remains to show that CDNL-PROJECTION$(\Pi, P, 0)$ is terminating. By the same argument as in the proof of Theorem 4.16, we have that CDNL-PROJECTION$(\Pi, P, 0)$ admits only finitely many backjumps in Line 12 of Algorithm 4.7. Likewise, only finitely many backtracks can be performed in Line 17 or 27 because the complement $\overline{\sigma_d}$ of a former (fake) decision entry $\sigma_d \in \mathbf{A}^P$ is assigned immediately afterwards, that is, in Line 18 or 28, respectively, and $dlevel(\overline{\sigma_d}) = dlevel(\sigma_d) - 1$ holds in view of Line 35–36, 15–16,

and 25–26. Furthermore, if an enumerated solution $\mathbf{A}$ for $\Delta_\Pi \cup \Lambda_\Pi$ does not immediately lead to either termination in Line 22 or a backtrack in Line 27, after incrementing bl in Line 30, an entry $\sigma_d \in \mathbf{A}^P$ such that $dlevel(\sigma_d) = bl$ is reassigned in Line 37, so that only $|P \cap atom(\Pi)|$ many solutions can be enumerated in a row without decreasing bl upon backtracking. Along with the fact that $\mathbf{A}$ is strictly extended in Line 41, so that either a backjump (in Line 12) or a backtrack (in Line 17 or 27, provided that the termination condition in Line 22 does not apply beforehand) is inevitable within a quadratic number of iterations of the loop in Line 4–41, we conclude that CDNL-PROJECTION$(\Pi, P, 0)$ eventually terminates in Line 7 or 22 of Algorithm 4.7. □

We have thus proven all formal results presented in Chapter 4.

List of Figures

List of Tables

List of Algorithms

Index

Bibliography

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] C. Anger, M. Gebser, T. Janhunen, and T. Schaub. What's a head without a body? In Brewka et al. [29], pages 769–770.

[3] C. Anger, M. Gebser, T. Linke, A. Neumann, and T. Schaub. The `nomore++` approach to answer set solving. In G. Sutcliffe and A. Voronkov, editors, *Proceedings of the Twelfth International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, volume 3835 of *Lecture Notes in Artificial Intelligence*, pages 95–109. Springer-Verlag, 2005.

[4] C. Anger, M. Gebser, and T. Schaub. Approaching the core of unfounded sets. In J. Dix and A. Hunter, editors, *Proceedings of the Eleventh International Workshop on Nonmonotonic Reasoning (NMR'06)*, number IFI-06-04 in Institute for Informatics, Clausthal University of Technology, Technical Report Series, pages 58–66, 2006.

[5] C. Anger, K. Konczak, T. Linke, and T. Schaub. A glimpse of answer set programming. *Künstliche Intelligenz*, 19(1):12–17, 2005.

[6] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann Publishers, 1987.

[7] G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In Boutilier [24], pages 399–404.

[8] Y. Babovich, E. Erdem, and V. Lifschitz. Fages' theorem and answer set programming. In C. Baral and M. Truszczyński, editors, *Proceedings of the Eighth International Workshop on Nonmonotonic Reasoning (NMR'00)*, 2000.

[9] Y. Babovich and V. Lifschitz. Computing answer sets using program completion. `http://www.cs.utexas.edu/users/tag/cmodels/cmodels-1.ps`, 2003. Unpublished draft.

[10] F. Bacchus, S. Dalmao, and T. Pitassi. Solving #SAT and Bayesian inference with backtracking search. *Journal of Artificial Intelligence Research*, 34:391–442, 2009.

[11] M. Balduccini. Industrial-size scheduling with ASP+CP. In Delgrande and Faber [46], pages 284–296.

[12] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[13] C. Baral, G. Brewka, and J. Schlipf, editors. *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, volume 4483 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2007.

[14] C. Baral, G. Greco, N. Leone, and G. Terracina, editors. *Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05)*, volume 3662 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2005.

[15] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability modulo theories. In Biere et al. [21], chapter 26, pages 825–885.

[16] R. Bayardo and J. Pehoushek. Counting models using connected components. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00)*, pages 157–162. AAAI Press/MIT Press, 2000.

[17] R. Bayardo and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI'97)*, pages 203–208. AAAI Press/MIT Press, 1997.

[18] P. Beame, H. Kautz, and A. Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, 2004.

[19] P. Beame and T. Pitassi. Propositional proof complexity: Past, present, and future. *Bulletin of the European Association for Theoretical Computer Science*, 65:66–89, 1998.

[20] A. Biere. PicoSAT essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.

[21] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[22] G. Boenn, M. Brain, M. De Vos, and J. Fitch. Automatic composition of melodic and harmonic music by answer set programming. In Garcia de la Banda and Pontelli [77], pages 160–174.

[23] P. Bonatti. Resolution for skeptical stable model semantics. *Journal of Automated Reasoning*, 27(4):391–421, 2001.

[24] C. Boutilier, editor. *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI'09)*. AAAI Press/MIT Press, 2009.

[25] M. Brain, T. Crick, M. De Vos, and J. Fitch. TOAST: Applying answer set programming to superoptimisation. In Etalle and Truszczyński [63], pages 270–284.

[26] M. Brain and M. De Vos. The significance of memory costs in answer set solver implementation. *Journal of Logic and Computation*, 19(4):615–641, 2009.

[27] M. Brain, M. Gebser, J. Pührer, T. Schaub, H. Tompits, and S. Woltran. Debugging ASP programs by means of ASP. In Baral et al. [13], pages 31–43.

[28] M. Brain, M. Gebser, J. Pührer, T. Schaub, H. Tompits, and S. Woltran. That is illogical captain! — the debugging support tool spock for answer-set programs: System description. In De Vos and Schaub [44], pages 71–85.

[29] G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, editors. *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI'06)*. IOS Press, 2006.

[30] G. Brewka and J. Lang, editors. *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*. AAAI Press, 2008.

[31] G. Brewka, I. Niemelä, and M. Truszczyński. Nonmonotonic reasoning. In Lifschitz et al. [166], chapter 6, pages 239–284.

[32] F. Calimeri, W. Faber, G. Pfeifer, and N. Leone. Pruning operators for disjunctive logic programming systems. *Fundamenta Informaticae*, 71(2-3):183–214, 2006.

[33] X. Chen, J. Ji, and F. Lin. Computing loops with at most one external support rule. In Brewka and Lang [30], pages 401–410.

[34] X. Chen, J. Ji, and F. Lin. Computing loops with at most one external support rule for disjunctive logic programs. In Erdem et al. [61], pages 130–144.

[35] Y. Chen, F. Lin, Y. Wang, and M. Zhang. First-order loop formulas for normal logic programs. In P. Doherty, J. Mylopoulos, and C. Welty, editors, *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 298–307. AAAI Press, 2006.

[36] K. Claessen and N. Sörensson. New techniques that improve MACE-style finite model finding. In P. Baumgartner and C. Fermüller, editors, *Proceedings of the Workshop on Model Computation — Principles, Algorithms, Applications (MODEL'03)*, 2003.

[37] K. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[38] Clasp. `http://www.cs.uni-potsdam.de/clasp`.

[39] S. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

[40] M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors. *Handbook of Tableau Methods*. Kluwer Academic Publishers, 1999.

[41] A. Darwiche and K. Pipatsrisawat. Complete algorithms. In Biere et al. [21], chapter 3, pages 99–130.

[42] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[43] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[44] M. De Vos and T. Schaub, editors. *Proceedings of the First Workshop on Software Engineering for Answer Set Programming (SEA'07)*, number CSBU-2007-05 in Department of Computer Science, University of Bath, Technical Report Series, 2007.

[45] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.

[46] J. Delgrande and W. Faber, editors. *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, volume 6645 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2011.

[47] M. Denecker, J. Vennekens, S. Bond, M. Gebser, and M. Truszczyński. The second answer set programming competition. In Erdem et al. [61], pages 637–654.

[48] N. Dershowitz, Z. Hanna, and A. Nadel. Towards a better understanding of the functionality of a conflict-driven SAT solver. In Marques-Silva and Sakallah [180], pages 287–293.

[49] J. Dix, U. Furbach, and I. Niemelä. Nonmonotonic reasoning: Towards efficient calculi and implementations. In J. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1241–1354. Elsevier and MIT Press, 2001.

[50] H. Dixon, M. Ginsberg, and A. Parkes. Generalizing Boolean satisfiability I: Background and survey of existing work. *Journal of Artificial Intelligence Research*, 21:193–243, 2004.

[51] A. Dovier and E. Pontelli, editors. *A 25-Year Perspective on Logic Programming*, volume 6125 of *Lecture Notes in Computer Science*. Springer-Verlag, 2010.

[52] C. Drescher, M. Gebser, T. Grote, B. Kaufmann, A. König, M. Ostrowski, and T. Schaub. Conflict-driven disjunctive answer set solving. In Brewka and Lang [30], pages 422–432.

[53] C. Drescher, M. Gebser, B. Kaufmann, and T. Schaub. Heuristics in conflict resolution. In M. Pagnucco and M. Thielscher, editors, *Proceedings of the Twelfth International Workshop on Nonmonotonic Reasoning (NMR'08)*, number UNSW-CSE-TR-0819 in School of Computer Science and Engineering, University of New South Wales, Technical Report Series, pages 141–149, 2008.

[54] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In F. Bacchus and T. Walsh, editors, *Proceedings of the Eigth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag, 2005.

[55] N. Eén and N. Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4), 2003.

[56] N. Eén and N. Sörensson. An extensible SAT-solver. In Giunchiglia and Tacchella [126], pages 502–518.

[57] T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995.

[58] T. Eiter, H. Tompits, and S. Woltran. On solution correspondences in answer set programming. In Kaelbling and Saffiotti [148], pages 97–102.

[59] E. Ellguth, M. Gebser, M. Gusowski, R. Kaminski, B. Kaufmann, S. Liske, T. Schaub, L. Schneidenbach, and B. Schnor. A simple distributed conflict-driven answer set solver. In Erdem et al. [61], pages 490–495.

[60] E. Erdem and V. Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 3(4-5):499–518, 2003.

[61] E. Erdem, F. Lin, and T. Schaub, editors. *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, volume 5753 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2009.

[62] E. Erdem and F. Türe. Efficient haplotype inference with answer set programming. In Fox and Gomes [75], pages 436–441.

[63] S. Etalle and M. Truszczyński, editors. *Proceedings of the Twenty-second International Conference on Logic Programming (ICLP'06)*, volume 4079 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006.

[64] W. Faber. *Enhancing Efficiency and Expressiveness in Answer Set Programming Systems*. Dissertation, Technische Universität Wien, 2002.

[65] W. Faber. Unfounded sets for disjunctive logic programs with arbitrary aggregates. In Baral et al. [14], pages 40–52.

[66] W. Faber, G. Pfeifer, and N. Leone. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2011.

[67] W. Faber, G. Pfeifer, N. Leone, T. Dell'Armi, and G. Ielpa. Design and implementation of aggregate functions in the DLV system. *Theory and Practice of Logic Programming*, 8(5-6):545–580, 2008.

[68] F. Fages. Consistency of Clark's completion and the existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.

[69] P. Ferraris. Answer sets for propositional theories. In Baral et al. [14], pages 119–131.

[70] P. Ferraris, J. Lee, and V. Lifschitz. A generalization of the Lin-Zhao theorem. *Annals of Mathematics and Artificial Intelligence*, 47(1-2):79–101, 2006.

[71] P. Ferraris, J. Lee, and V. Lifschitz. A new perspective on stable models. In Veloso [217], pages 372–379.

[72] P. Ferraris and V. Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5(1-2):45–74, 2005.

[73] M. Fitting. Tableaux for logic programming. *Journal of Automated Reasoning*, 13(2):175–188, 1994.

[74] M. Fitting. Fixpoint semantics for logic programming: A survey. *Theoretical Computer Science*, 278(1-2):25–51, 2002.

[75] D. Fox and C. Gomes, editors. *Proceedings of the Twenty-third National Conference on Artificial Intelligence (AAAI'08)*. AAAI Press, 2008.

[76] J. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania, 1995.

[77] M. Garcia de la Banda and E. Pontelli, editors. *Proceedings of the Twenty-fourth International Conference on Logic Programming (ICLP'08)*, volume 5366 of *Lecture Notes in Computer Science*. Springer-Verlag, 2008.

[78] M. Gebser, T. Grote, R. Kaminski, and T. Schaub. Reactive answer set programming. In Delgrande and Faber [46], pages 54–66.

[79] M. Gebser, T. Grote, and T. Schaub. Coala: A compiler from action languages to ASP. In Janhunen and Niemelä [138], pages 360–364.

[80] M. Gebser, C. Guziolowski, M. Ivanchev, T. Schaub, A. Siegel, S. Thiele, and P. Veber. Repair and prediction (under inconsistency) in large biological networks with answer set programming. In F. Lin and U. Sattler, editors, *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning (KR'10)*, pages 497–507. AAAI Press, 2010.

[81] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):105–124, 2011.

[82] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo`. [198]

[83] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and S. Thiele. Engineering an incremental ASP solver. In Garcia de la Banda and Pontelli [77], pages 190–205.

[84] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. On the implementation of weight constraint rules in conflict-driven ASP solvers. In Hill and Warren [132], pages 250–264.

[85] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Multi-criteria optimization in answer set programming. In J. Gallagher and M. Gelfond, editors, *Technical Communications of the Twenty-seventh International Conference on Logic Programming (ICLP'11)*, volume 11 of *Leibniz International Proceedings in Informatics*, pages 1–10. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2011.

[86] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Multi-criteria optimization in ASP and its application to Linux package configuration. In D. Le Berre and A. Van Gelder, editors, *Proceedings of the Second Workshop on Pragmatics of SAT (PoS'11)*, 2011. Submitted for post-proceedings.

[87] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, M. Schneider, and S. Ziller. A portfolio solver for answer set programming: Preliminary report. In Delgrande and Faber [46], pages 352–357.

[88] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, and B. Schnor. Cluster-based ASP solving with *claspar*. In Delgrande and Faber [46], pages 364–369.

[89] M. Gebser, R. Kaminski, M. Knecht, and T. Schaub. plasp: A prototype for PDDL-based planning in ASP. In Delgrande and Faber [46], pages 358–363.

[90] M. Gebser, R. Kaminski, A. König, and T. Schaub. Advances in *gringo* series 3. In Delgrande and Faber [46], pages 345–351.

[91] M. Gebser, R. Kaminski, M. Ostrowski, T. Schaub, and S. Thiele. On the input language of ASP grounder gringo. In Erdem et al. [61], pages 502–508.

[92] M. Gebser, R. Kaminski, and T. Schaub. Complex optimization in answer set programming. *Theory and Practice of Logic Programming, Twenty-seventh International Conference on Logic Programming (ICLP'11) Special Issue*, 11(4-5):821–839, 2011.

[93] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. In Baral et al. [13], pages 260–265.

[94] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set enumeration. In Baral et al. [13], pages 136–148.

[95] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In Veloso [217], pages 386–392.

[96] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Advanced preprocessing for answer set solving. In M. Ghallab, C. Spyropoulos, N. Fakotakis, and N. Avouris, editors, *Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, pages 15–19. IOS Press, 2008.

[97] M. Gebser, B. Kaufmann, and T. Schaub. The conflict-driven answer set solver clasp: Progress report. In Erdem et al. [61], pages 509–514.

[98] M. Gebser, B. Kaufmann, and T. Schaub. Solution enumeration for projected Boolean search problems. In W. van Hoeve and J. Hooker, editors, *Proceedings of the Sixth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09)*, volume 5547 of *Lecture Notes in Computer Science*, pages 71–86. Springer-Verlag, 2009.

[99] M. Gebser, B. Kaufmann, and T. Schaub. Conflict-driven answer set solving: From theory to practice. *Artificial Intelligence*. To appear.

[100] M. Gebser, A. König, T. Schaub, S. Thiele, and P. Veber. The BioASP library: ASP solutions for systems biology. In E. Grégoire, editor, *Proceedings of the Twenty-second IEEE International Conference on Tools with Artificial Intelligence (ICTAI'10)*, pages 383–389. IEEE Computer Society, 2010.

[101] M. Gebser, J. Lee, and Y. Lierler. Elementary sets for logic programs. In Gil and Mooney [121], pages 244–249.

[102] M. Gebser, J. Lee, and Y. Lierler. Head-elementary-set-free logic programs. In Baral et al. [13], pages 149–161.

[103] M. Gebser, J. Lee, and Y. Lierler. On elementary loops of logic programs. *Theory and Practice of Logic Programming*. To appear.

[104] M. Gebser, L. Liu, G. Namasivayam, A. Neumann, T. Schaub, and M. Truszczyński. The first answer set programming system competition. In Baral et al. [13], pages 3–17.

[105] M. Gebser, M. Ostrowski, and T. Schaub. Constraint answer set solving. In Hill and Warren [132], pages 235–249.

[106] M. Gebser, J. Pührer, T. Schaub, and H. Tompits. A meta-programming technique for debugging answer-set programs. In Fox and Gomes [75], pages 448–453.

[107] M. Gebser, O. Sabuncu, and T. Schaub. An incremental answer set programming based system for finite model computation. In Janhunen and Niemelä [138], pages 169–181.

[108] M. Gebser, O. Sabuncu, and T. Schaub. An incremental answer set programming based system for finite model computation. *AI Communications*, 24(2):195–212, 2011.

[109] M. Gebser and T. Schaub. Loops: Relevant or redundant? In Baral et al. [14], pages 53–65.

[110] M. Gebser and T. Schaub. Characterizing ASP inferences by unit propagation. In E. Giunchiglia, V. Marek, D. Mitchell, and E. Ternovska, editors, *Proceedings of the First International Workshop on Search and Logic: Answer Set Programming and SAT (LaSh'06)*, pages 41–56, 2006.

[111] M. Gebser and T. Schaub. Tableau calculi for answer set programming. In Etalle and Truszczyński [63], pages 11–25.

[112] M. Gebser and T. Schaub. Generic tableaux for answer set programming. In V. Dahl and I. Niemelä, editors, *Proceedings of the Twenty-third International Conference on Logic Programming (ICLP'07)*, volume 4670 of *Lecture Notes in Computer Science*, pages 119–133. Springer-Verlag, 2007.

[113] M. Gebser and T. Schaub. Tableau calculi for logic programs under answer set semantics. *ACM Transactions on Computational Logic*. To appear.

[114] M. Gebser, T. Schaub, and S. Thiele. Gringo: A new grounder for answer set programming. In Baral et al. [13], pages 266–271.

[115] M. Gebser, T. Schaub, S. Thiele, and P. Veber. Detecting inconsistencies in large biological networks with answer set programming. *Theory and Practice of Logic Programming*, 11(2-3):323–360, 2011.

[116] M. Gebser, T. Schaub, H. Tompits, and S. Woltran. Alternative characterizations for program equivalence under answer-set semantics based on unfounded sets. In S. Hartmann and G. Kern-Isberner, editors, *Proceedings of the Fifth International Symposium on Foundations of Information and Knowledge Systems (FoIKS'08)*, volume 4932 of *Lecture Notes in Computer Science*, pages 24–41. Springer-Verlag, 2008.

[117] M. Gelfond. Answer sets. In Lifschitz et al. [166], chapter 7, pages 285–316.

[118] M. Gelfond and N. Leone. Logic programming and knowledge representation — the A-Prolog perspective. *Artificial Intelligence*, 138(1-2):3–38, 2002.

[119] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, pages 1070–1080. MIT Press, 1988.

[120] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[121] Y. Gil and R. Mooney, editors. *Proceedings of the Twenty-first National Conference on Artificial Intelligence (AAAI'06)*. AAAI Press, 2006.

[122] E. Giunchiglia, N. Leone, and M. Maratea. On the relation among answer set solvers. *Annals of Mathematics and Artificial Intelligence*, 53(1-4):169–204, 2008.

[123] E. Giunchiglia, Y. Lierler, and M. Maratea. Answer set programming based on propositional satisfiability. *Journal of Automated Reasoning*, 36(4):345–377, 2006.

[124] E. Giunchiglia and M. Maratea. On the relation between answer set and SAT procedures (or, between cmodels and smodels). In M. Gabbrielli and G. Gupta, editors, *Proceedings of the Twenty-first International Conference on Logic Programming (ICLP'05)*, volume 3668 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 2005.

[125] E. Giunchiglia and M. Maratea. Solving optimization problems with DLL. In Brewka et al. [29], pages 377–381.

[126] E. Giunchiglia and A. Tacchella, editors. *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, volume 2919 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[127] E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT solver. In *Proceedings of the Fifth Conference on Design, Automation and Test in Europe (DATE'02)*, pages 142–149. IEEE Computer Society, 2002.

[128] C. Gomes, A. Sabharwal, and B. Selman. Model counting. In Biere et al. [21], chapter 20, pages 633–654.

[129] G. Grasso, S. Iiritano, N. Leone, and F. Ricca. Some DLV applications for knowledge management. In Erdem et al. [61], pages 591–597.

[130] O. Grumberg, A. Schuster, and A. Yadgar. Memory efficient all-solutions SAT solver and its application for reachability analysis. In A. Hu and A. Martin, editors, *Proceedings of the Fifth International Conference on Formal Methods in Computer-Aided Design (FMCAD'04)*, volume 3312 of *Lecture Notes in Computer Science*, pages 275–289. Springer-Verlag, 2004.

[131] R. Hähnle. Tableaux and related methods. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 100–178. Elsevier and MIT Press, 2001.

[132] P. Hill and D. Warren, editors. *Proceedings of the Twenty-fifth International Conference on Logic Programming (ICLP'09)*, volume 5649 of *Lecture Notes in Computer Science*. Springer-Verlag, 2009.

[133] J. Huang. The effect of restarts on the efficiency of clause learning. In Veloso [217], pages 2318–2323.

[134] H. Ishebabi, P. Mahr, C. Bobda, M. Gebser, and T. Schaub. Answer set vs integer linear programming for automatic synthesis of multiprocessor systems from real-time parallel programs. *Journal of Reconfigurable Computing*, 2009. Article ID 863630.

[135] H. Ishebabi, P. Mahr, C. Bobda, M. Gebser, and T. Schaub. Application of ASP for automatic synthesis of flexible multiprocessor systems from parallel programs. In Erdem et al. [61], pages 598–603.

[136] T. Janhunen. Some (in)translatability results for normal logic programs and propositional theories. *Journal of Applied Non-Classical Logics*, 16(1-2):35–86, 2006.

[137] T. Janhunen, G. Liu, and I. Niemelä. Tight integration of non-ground answer set programming and satisfiability modulo theories. In P. Cabalar, D. Mitchell, D. Pearce, and E. Ternovska, editors, *Proceedings of the First Workshop on Grounding and Transformation for Theories with Variables (GTTV'11)*, pages 1–13, 2011.

[138] T. Janhunen and I. Niemelä, editors. *Proceedings of the Twelfth European Conference on Logics in Artificial Intelligence (JELIA'10)*, volume 6341 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2010.

[139] T. Janhunen and I. Niemelä. Compact translations of non-disjunctive answer set programs to propositional clauses. In M. Balduccini and T. Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, volume 6565 of *Lecture Notes in Computer Science*, pages 111–130. Springer-Verlag, 2011.

[140] T. Janhunen, I. Niemelä, D. Seipel, P. Simons, and J. You. Unfolding partiality and disjunctions in stable model semantics. *ACM Transactions on Computational Logic*, 7(1):1–37, 2006.

[141] T. Janhunen, I. Niemelä, and M. Sevalnev. Computing stable models via reductions to difference logic. In Erdem et al. [61], pages 142–154.

[142] M. Järvisalo. Itemset mining as a challenge application for answer set enumeration. In Delgrande and Faber [46], pages 304–310.

[143] M. Järvisalo and T. Junttila. Limitations of restricted branching in clause learning. *Constraints*, 14(3):325–356, 2009.

[144] M. Järvisalo, T. Junttila, and I. Niemelä. Unrestricted vs restricted cut in a tableau method for Boolean circuits. *Annals of Mathematics and Artificial Intelligence*, 44(4):373–399, 2005.

[145] M. Järvisalo and E. Oikarinen. Extended ASP tableaux and rule redundancy in normal logic programs. *Theory and Practice of Logic Programming*, 8(5-6):691–716, 2008.

[146] H. Jin, H. Han, and F. Somenzi. Efficient conflict analysis for finding all satisfying assignments of a Boolean circuit. In N. Halbwachs and L. Zuck, editors, *Proceedings of the Eleventh International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05)*, volume 3440 of *Lecture Notes in Computer Science*, pages 287–300. Springer-Verlag, 2005.

[147] T. Junttila and P. Kaski. Exact cover via satisfiability: An empirical study. In D. Cohen, editor, *Proceedings of the Sixteenth International Conference on Principles and Practice of Constraint Programming (CP'10)*, volume 6308 of *Lecture Notes in Computer Science*, pages 297–304. Springer-Verlag, 2010.

[148] L. Kaelbling and A. Saffiotti, editors. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI'05)*. Professional Book Center, 2005.

[149] G. Katsirelos. *Nogood Processing in CSPs*. PhD thesis, University of Toronto, 2008.

[150] G. Katsirelos and F. Bacchus. Generalized nogoods in CSPs. In M. Veloso and S. Kambhampati, editors, *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI'05)*, pages 390–396. AAAI Press, 2005.

[151] S. Khurshid, D. Marinov, I. Shlyakhter, and D. Jackson. A case for efficient solution enumeration. In Giunchiglia and Tacchella [126], pages 272–286.

[152] T. Kim, J. Lee, and R. Palla. Circumscriptive event calculus as answer set programming. In Boutilier [24], pages 823–829.

[153] H. Kleine Büning and X. Zhao, editors. *Proceedings of the Eleventh International Conference on Theory and Applications of Satisfiability Testing (SAT'08)*, volume 4996 of *Lecture Notes in Computer Science*. Springer-Verlag, 2008.

[154] K. Konczak, T. Linke, and T. Schaub. Graphs and colorings for answer set programming. *Theory and Practice of Logic Programming*, 6(1-2):61–106, 2006.

[155] S. Lahiri, R. Nieuwenhuis, and A. Oliveras. SMT techniques for fast predicate abstraction. In T. Ball and R. Jones, editors, *Proceedings of the Eighteenth International Conference on Computer Aided Verification (CAV'06)*, volume 4144 of *Lecture Notes in Computer Science*, pages 424–437. Springer-Verlag, 2006.

[156] J. Lee. A model-theoretic counterpart of loop formulas. In Kaelbling and Saffiotti [148], pages 503–508.

[157] J. Lee and Y. Meng. On loop formulas with variables. In Brewka and Lang [30], pages 444–453.

[158] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[159] N. Leone, P. Rullo, and F. Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Information and Computation*, 135(2):69–112, 1997.

[160] Y. Lierler. Abstract answer set solvers with learning. *Theory and Practice of Logic Programming*. To appear.

[161] V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

[162] V. Lifschitz. Twelve definitions of a stable model. In Garcia de la Banda and Pontelli [77], pages 37–51.

[163] V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.

[164] V. Lifschitz and A. Razborov. Why are there so many loop formulas? *ACM Transactions on Computational Logic*, 7(2):261–268, 2006.

[165] V. Lifschitz, L. Tang, and H. Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389, 1999.

[166] V. Lifschitz, F. van Harmelen, and B. Porter, editors. *Handbook of Knowledge Representation*. Elsevier, 2008.

[167] F. Lin and Y. Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157(1-2):115–137, 2004.

[168] Z. Lin, Y. Zhang, and H. Hernandez. Fast SAT-based answer set solver. In Gil and Mooney [121], pages 92–97.

[169] T. Linke, C. Anger, and K. Konczak. More on `nomore`. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Proceedings of the Eighth European Conference on Logics in Artificial Intelligence (JELIA'02)*, volume 2424 of *Lecture Notes in Computer Science*, pages 468–480. Springer-Verlag, 2002.

[170] G. Liu and J. You. Level mapping induced loop formulas for weight constraint and aggregate logic programs. *Fundamenta Informaticae*, 101(3):237–255, 2010.

[171] L. Liu and M. Truszczyński. Properties and applications of programs with monotone and convex constraints. *Journal of Artificial Intelligence Research*, 27:299–334, 2006.

[172] J. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.

[173] V. Marek and J. Remmel. On the continuity of Gelfond-Lifschitz operator and other applications of proof-theory in ASP. In Garcia de la Banda and Pontelli [77], pages 223–237.

[174] V. Marek and V. Subrahmanian. The relationship between stable, supported, default and autoepistemic semantics for general logic programs. *Theoretical Computer Science*, 103(2):365–386, 1992.

[175] V. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38(3):588–619, 1991.

[176] V. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In K. Apt, W. Marek, M. Truszczyński, and D. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[177] M. Mariën, J. Wittocx, M. Denecker, and M. Bruynooghe. SAT(ID): Satisfiability of propositional logic extended with inductive definitions. In Kleine Büning and Zhao [153], pages 211–224.

[178] J. Marques-Silva, I. Lynce, and S. Malik. Conflict-driven clause learning SAT solvers. In Biere et al. [21], chapter 4, pages 131–153.

[179] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[180] J. Marques-Silva and K. Sakallah, editors. *Proceedings of the Tenth International Conference on Theory and Applications of Satisfiability Testing (SAT'07)*, volume 4501 of *Lecture Notes in Computer Science*. Springer-Verlag, 2007.

[181] V. Mellarkod, M. Gelfond, and Y. Zhang. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence*, 53(1-4):251–287, 2008.

[182] A. Mileo, D. Merico, and R. Bisiani. A logic programming approach to home monitoring for risk prevention in assisted living. In Garcia de la Banda and Pontelli [77], pages 145–159.

[183] D. Mitchell. A SAT solver primer. *Bulletin of the European Association for Theoretical Computer Science*, 85:112–133, 2005.

[184] A. Morgado and J. Marques-Silva. Good learning and implicit model enumeration. In *Proceedings of the Seventeenth IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)*, pages 131–136. IEEE Computer Society, 2005.

[185] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Thirty-eighth Conference on Design Automation (DAC'01)*, pages 530–535. ACM Press, 2001.

[186] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):241–273, 1999.

[187] I. Niemelä. Stable models and difference logic. *Annals of Mathematics and Artificial Intelligence*, 53(1-4):313–329, 2008.

[188] R. Nieuwenhuis and A. Oliveras. DPLL(T) with exhaustive theory propagation and its application to difference logic. In K. Etessami and S. Rajamani, editors, *Proceedings of the Seventeenth International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 321–334. Springer-Verlag, 2005.

[189] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.

[190] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An A-Prolog decision support system for the space shuttle. In I. Ramakrishnan, editor, *Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages (PADL'01)*, volume 1990 of *Lecture Notes in Computer Science*, pages 169–183. Springer-Verlag, 2001.

[191] N. Olivetti. Tableaux for nonmonotonic logics. In D'Agostino et al. [40], pages 469–528.

[192] D. Pearce. A new logical characterisation of stable models and answer sets. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Proceedings of the Sixth Workshop on Non-Monotonic Extensions of Logic Programming (NMELP'96)*, volume 1216 of *Lecture Notes in Computer Science*, pages 57–70. Springer-Verlag, 1996.

[193] D. Pearce, I. de Guzmán, and A. Valverde. A tableau calculus for equilibrium entailment. In R. Dyckhoff, editor, *Proceedings of the Ninth International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'00)*, volume 1847 of *Lecture Notes in Computer Science*, pages 352–367. Springer-Verlag, 2000.

[194] D. Pearce and A. Valverde. A first order nonmonotonic extension of constructive logic. *Studia Logica*, 30(2-3):321–346, 2005.

[195] K. Pipatsrisawat and A. Darwiche. A lightweight component caching scheme for satisfiability solvers. In Marques-Silva and Sakallah [180], pages 294–299.

[196] K. Pipatsrisawat and A. Darwiche. A new clause learning scheme for efficient unsatisfiability proofs. In Fox and Gomes [75], pages 1481–1484.

[197] K. Pipatsrisawat and A. Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence*, 175(2):512–525, 2011.

[198] Potassco. `http://potassco.sourceforge.net`.

[199] P. Purdom. A transitive closure algorithm. *BIT Numerical Mathematics*, 10:76–94, 1970.

[200] F. Ricca, W. Faber, and N. Leone. A backjumping technique for disjunctive logic programming. *AI Communications*, 19(2):155–172, 2006.

[201] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

[202] O. Roussel and V. Manquinho. Pseudo-Boolean and cardinality constraints. In Biere et al. [21], chapter 22, pages 695–733.

[203] L. Ryan. Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University, 2004.

[204] V. Ryvchin and O. Strichman. Local restarts. In Kleine Büning and Zhao [153], pages 271–276.

[205] K. Sakallah. Symmetry and satisfiability. In Biere et al. [21], chapter 10, pages 289–338.

[206] J. Schlipf. The expressive powers of the logic programming semantics. *Journal of Computer and System Sciences*, 51:64–86, 1995.

[207] L. Schneidenbach, B. Schnor, M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Experiences running a parallel answer set solver on Blue Gene. In M. Ropo, J. Westerholm, and J. Dongarra, editors, *Proceedings of the Sixteenth European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (PVM/MPI'09)*, volume 5759 of *Lecture Notes in Computer Science*, pages 64–72. Springer-Verlag, 2009.

[208] P. Simons. *Extending and Implementing the Stable Model Semantics*. Dissertation, Helsinki University of Technology, 2000.

[209] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234, 2002.

[210] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In G. Gupta, editor, *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages (PADL'99)*, volume 1551 of *Lecture Notes in Computer Science*, pages 305–319. Springer-Verlag, 1999.

[211] T. Syrjänen. Lparse 1.0 user's manual. `http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz`.

[212] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[213] M. Thielscher. Answer set programming for single-player games in general game playing. In Hill and Warren [132], pages 327–341.

[214] M. Truszczyński. Comments on modeling languages for answer-set programming. In De Vos and Schaub [44], pages 3–11.

[215] J. Ullman. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, 1988.

[216] A. Van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[217] M. Veloso, editor. *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*. AAAI Press/MIT Press, 2007.

[218] J. Ward and J. Schlipf. Answer set programming with clause learning. In V. Lifschitz and I. Niemelä, editors, *Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04)*, volume 2923 of *Lecture Notes in Artificial Intelligence*, pages 302–313. Springer-Verlag, 2004.

[219] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'01)*, pages 279–285, 2001.

[220] L. Zhang and S. Malik. Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. In *Proceedings of the Sixth Conference on Design, Automation and Test in Europe (DATE'03)*, pages 10880–10885. IEEE Computer Society, 2003.